University of California

Los Angeles

Query Relaxation for XML Model

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Dongwon Lee

2002

The dissertation of Dongwon Lee is approved.

Jonathan Furner

D. Stott Parker, Jr.

Carlo Zaniolo

Wesley W. Chu, Committee Chair

University of California, Los Angeles

2002

To my parents,

Eung-Ok Lee and *Hyang-Dae Kim*,

who always believed in me,
even when everyone else was in doubt.

And to my family,
Jean Oh and *Sylvie Lee*,
who made all of this possible,
for their endless encouragement and patience.

No discovery of mine has made, or is likely to make, directly or indirectly, for good or ill, the least difference to the amenity of the world.

— G. H. Hardy

Table of Contents

List of Figures

List of Tables

Acknowledgments

Writing this dissertation marks the happy conclusion of the journey that I started many years ago. Throughout the journey, I greatly benefited from the support and companionship of many people.

First and foremost, I would like to thank my advisor, Professor Wesley W. Chu, for providing the guidance and feedback on the research problems and issues studied in this dissertation. As I retrospect, I realize that I have learned so many things from him, technical or non-technical, directly or indirectly, that practically shaped who I am now as a scholar. I would also like to thank the other members of my Ph.D committee, Professor D. Stott Parker, Professor Carlo Zaniolo, and Professor Jonathan Further. All the members of my committee have in many ways shaped my education and have provided insightful feedback concerning this research. I must also admit that I learned a lot from Professor Carlo Zaniolo's classes, despite their offerings in the napping mid-afternoon.

Professor H. V. Jagadish at U. Michigan, Ann Arbor and Dr. Divesh Srivastava at AT&T Labs – Research are responsible for my decision to study Database, although they would never realize how much influential they were to me when we worked together at Bell Labs from 1995 to 1997. I especially like to thank Dr. Divesh Srivastava for showing me how to find research problems, how to approach them to solve, and how to enjoy life after graduate school. My dear friends from Korea University, Dr. Sang-Zoo Lee and Sung-Mi Cho, helped me maintain sanity during the Ph.D program by way of occasional emails and phone calls. Good friends from Columbia University, Jae-Jun Hwang, Taejin Yoon, and Chang-Hee Kim, have always been tremendous comforts to me, for which I deeply appreciate.

I would also like to thank many colleagues at UCLA, whom I have spent many hours, maybe too many, together for the last five years. Murali Mani has been an inspirational source for me in many ways, always asking me provocative and crucial questions and amazing me for his creative mind. Thanks to Wenlei Mao and Victor Liu, I was actually able to enjoy being in the crummy office at Boelter Hall. I have always admired Wenlei's clean logical thinking and Victor's sharp insights and deep understanding on the subjects. I only wish I had learned Chinese from them. Shanghyun Park and I had much fun together in playing tennis, eating snacks, chatting over coffees, and most of all, collaborating on research.

During my Ph.D study, I was fortunate to have chances to work closely with many bright students in the CoBase research group; Henry Chiu, Giovanni Giuffrida, Vivian Cheung, Tony Lee, Laura Chen, Qing Hua Zou, Akiko Nakaniwa. They all contributed to my dissertation one way or the other. Since early 2001, I have been exposed to diverse research topics through dbUCLA seminars. For that, I am indebted to those students who were involved in organizing the events together with me; Murali Mani, Victor Liu, Andrea Chu, Xia Yi, Fusheng Wang, Panayiotis Michael, Cindy Chen. I also wish to acknowledge a few names who had enriched my stay at UCLA; we had had so much fun over numerous lunches, arguing about virtually everything from politics to entertainment gossips; James Jinkyu Kim, Scott Seongwook Lee, Kyle Sangho Bae, Chang-Ki Choi, Bo-Kyung Choi, Yunjung Yi, Sungwook Lee, Heeyeol Yu, Ted Taekyung Kwon, Haejung Lim, Janghyuk J. Pyon.

Finally, and definitely not least, without the support and patience of my loving family, I would not be able to come this far. All of them equally deserve to sign this dissertation.

Vita

1969	Born, Taegu, Korea.
1990	Summer Intern, Samsung-Hewlett Packard Co., Seoul, Korea.
1992	Consulting Computer Instructor, Hyundai Co., Seoul, Korea.
1993	B.S. (Computer Science), Korea University, Seoul, Korea.
1994	Teaching Assistant, Computer Science Department, Columbia University, New York. Taught W4118 (Operating System) and W3824 (Computer Organization).
1995	M.S. (Computer Science), Columbia University, New York, NY.
1995–1996	Programmer, Database Research Department, AT&T Bell Labs. (Now Lucent Technology), Murray Hill, NJ.
1996–1997	Programmer, Database Research Department, AT&T Labs – Research, Murray Hill, NJ.
1997–2002	Research Assistant, Computer Science Department, UCLA, Los Angeles, CA.
2002	PH.D. (Computer Science), UCLA, Los Angeles, CA.

Publications

Murali Mani, **Dongwon Lee**, “Normal Forms for Regular Tree Grammars”, *Proc. VLDB Workshop on Efficiency and Effectiveness of XML Tools, and Techniques (EEXTT)*, Hong Kong, China, August 2002

Dongwon Lee, Murali Mani, Wesley W. Chu, “Effective Schema Conversions between XML and Relational Models”, *Proc. European Conf. on Artificial Intelligence (ECAI), Knowledge Transformation Workshop*, Lyon, France, July 2002

Dongwon Lee, Murali Mani, Frank Chiu, Wesley W. Chiu, “NeT & CoT: Inferring XML Schemas from Relational World”, *Proc. 18th IEEE Int'l Conf. on Data Engineering (ICDE)*, San Jose, CA, USA, February 2002

Dongwon Lee, Wesley W. Chu, “Towards Intelligent Semantic Caching for Web Sources”, *J. Intelligent Information System (JIIS)*, Vol. 17, No. 1, p 23–45, November 2001

Dongwon Lee, Wesley W. Chu, “CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema”, *J. Data & Knowledge Engineering (DKE)*, Vol. 39, No. 1, p 3–25, October 2001

Angela Bonifati, **Dongwon Lee**, “Technical Survey of XML Schema and Query Languages”, *Submitted to ACM Computing Survey*, September 2001

Dongwon Lee, Wenlei Mao, Henry Chiu, Wesley W. Chu, "Visual Trigger Rule Composition via Trigger-By-Example", *Submitted to J. Knowledge and Information Systems (KAIS)*, October 2001

Murali Mani, **Dongwon Lee**, Richard R. Muntz, "Semantic Data Modeling using XML Schemas", *Proc. 20th Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, November 2001

Makoto Murata, **Dongwon Lee**, Murali Mani, "Taxonomy of XML Schema Languages using Formal Language Theory", *Extreme Markup Languages*, Montreal, Canada, August 2001

Dongwon Lee, Murali Mani, Frank Chiu, Wesley W. Chu, "Nesting-based Relational-to-XML Schema Translation", *ACM SIGMOD Int'l Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, May 2001

Dongwon Lee, Wesley W. Chu, "Comparative Analysis of Six XML Schema Languages", *ACM SIGMOD Record*, Vol. 29, No. 3, September 2000

Dongwon Lee, Murali Mani, Makoto Murata, "Reasoning about XML Schema Languages using Formal Language Theory", *IBM Almaden Research Center, Technical Report*, RJ#10197, Log#95071, November 2000

Dongwon Lee, Wesley W. Chu, "Constraints-preserving Transformation from XML Document Type Definition to Relational Schema", *Proc. 19th Int'l Conf. on Conceptual Modeling (ER)*, Salt Lake City, Utah, October 2000

Dongwon Lee, Wenlei Mao, Wesley W. Chu, “TBE: Trigger-By-Example”, *Proc. 19th Int’l Conf. on Conceptual Modeling (ER)*, Salt Lake City, Utah, October 2000

Dongwon Lee, Wenlei Mao, Henry Chiu, Wesley W. Chu, “TBE: A Graphical Interface for Writing Trigger Rules in Active Databases”, *Proc. 5th IFIP 2.6 Working Conf. on Visual Database Systems (VDB)*, Fukuoka, Japan, May 2000

Sanghyun Park, **Dongwon Lee**, Wesley W. Chu, “Fast Retrieval of Similar Subsequences in Long Sequence Databases”, *Proc. 3rd IEEE Knowledge and Data Engineering Exchange Workshop (KDEX)*, Chicago, IL, November 1999

Dongwon Lee, Wesley W. Chu, “Semantic Caching via Query Matching for Web Sources”, *Proc. 8th ACM Int’l Conf. on Information and Knowledge Management (CIKM)*, Kansas City, MO, November 1999

Dongwon Lee, Wesley W. Chu, “Conjunctive Point Predicate-based Semantic Caching for Wrappers in Web Database”, *Proc. ACM CIKM Int’l Workshop on Web Information and Data Management (WIDM)*, Washington DC, USA, November 1998

Dongwon Lee, Divesh Srivastava, Dimitra Vista, “Generating Advanced Query Interfaces”, *Proc. 7th Int’l World Wide Web Conf. (WWW)*, Australia, April 1998

Abstract of the Dissertation

Query Relaxation for XML Model

by

Dongwon Lee

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2002

Professor Wesley W. Chu, Chair

This dissertation addresses mainly three issues needed to support query relaxation for XML model: framework formalization, extension of existing database techniques, and data conversion between XML and relational models.

XML (eXtensible Markup Language) is the new universal format for structured documents and data on the World Wide Web. As the Web becomes a major means of disseminating and sharing information and as the amount of XML data increases substantially, there are increased needs to manage and query such XML data in a novel yet efficient way. Especially, one of the useful query processing methods is known as *Query Relaxation*. Unlike relational databases where the schema is relatively small and fixed, XML model allows varied/missing structures and values, which make it difficult for users to ask questions precisely and completely. To address such problems, query relaxation technique enables systems to automatically weaken, when not satisfactory, the given user query to a less restricted form to permit *approximate* answers as well.

Therefore, the goal of our study is to investigate issues involved in supporting query relaxation for XML model. (1) We first present a formal framework where users can express the precise semantics and behaviors of query relaxation.

This framework can also be used as the basis for designing and implementing the eventual relaxation-enabled query language. (2) We then study a myriad of issues that are related to support query relaxation using native XML engines. Especially, we focus on the notion of similarity between XML data trees using tree edit distance and the issue of selectivity estimation of a set of relaxed XML queries. (3) Finally, we present issues involved in converting data between XML and relational models and propose three novel conversion algorithms that not only capture the original structures and values, but also well preserve the semantic constraints of the original schema. This is a necessary step to support query relaxation for XML model by way of using the mature relational database systems.

CHAPTER 1

Introduction

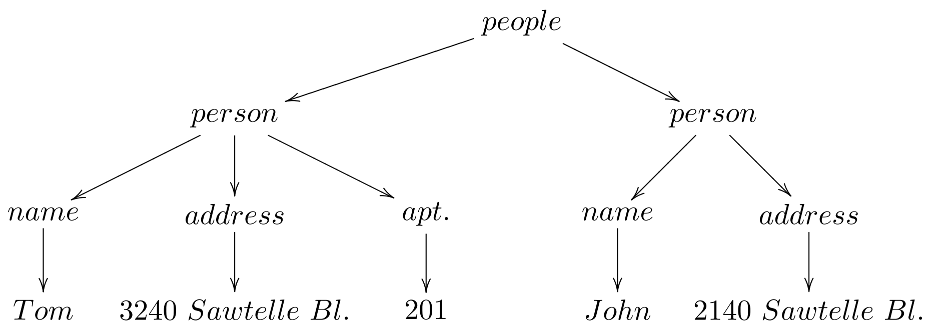
As the World Wide Web (Web) becomes a major means of disseminating and sharing information, there is an exponential increase in the amount of data in a web-compliant format such as HTML (HyperText Markup Language) [RHJ99] and XML (eXtensible Markup Language) [BPS00]. Especially, XML model is a novel textual representation of hierarchical (tree-like) data where a meaningful piece of data is bounded by matching starting and ending tags, such as `<name>` and `</name>`. Due to the simplicity of XML compared to SGML (Standard Generalized Markup Language) [ISO86] and its relatively powerful expressiveness compared to HTML, XML has become ubiquitous, and XML data has to be managed in databases.

In storing data in databases, the same data can be correctly captured in different models – relational or XML models. However, as illustrated in Figure 1.1, there are subtle differences. XML model representation can capture more detailed structural information (e.g., `person`) due to its hierarchical model and does not suffer from unnecessary null information (e.g., there is no `apt.` branch in the second `person` node). In relational model, data are represented in a "flat" structure where the only entities available are either tables or attributes. However, in XML model, a tree-like structure replaces the notion of tables, where leaf nodes are equivalent to the notion of attributes in relational model.

people

name	address	apt.
Tom	3240 Sawtelle Bl.	201
John	2140 Sepulveda Bl.	-

(a) Relational model



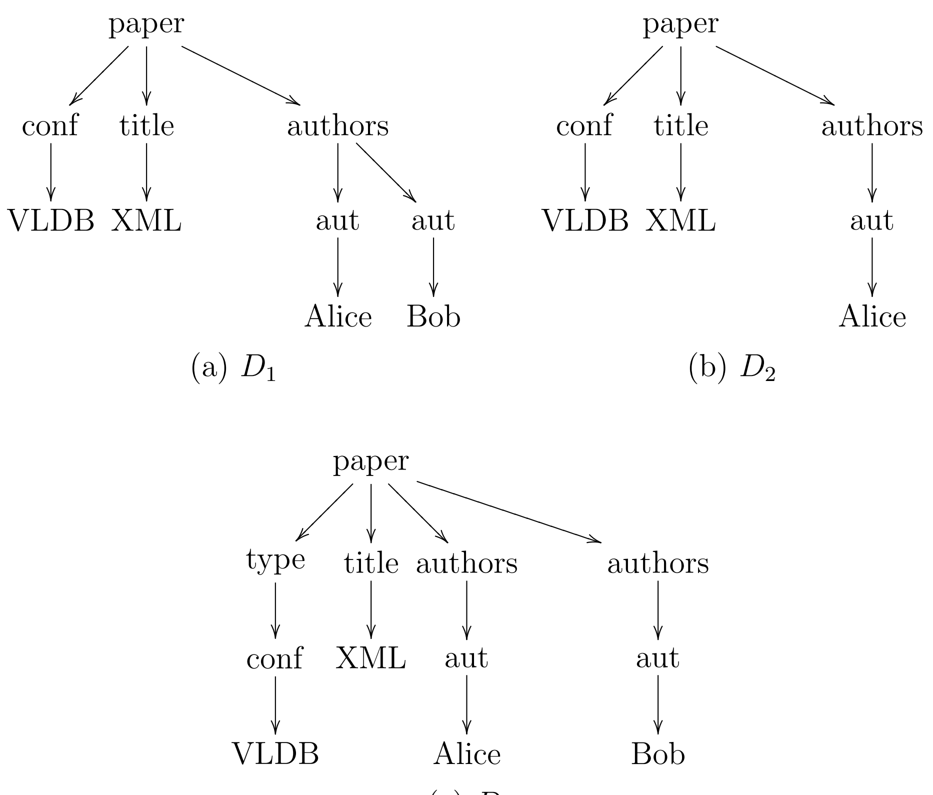
(b) XML model

Figure 1.1: Representing `people` in both relational and XML models.

1.1 Motivation

To cope with the tree-like structures in XML model, several XML-specific query languages have been proposed lately (e.g., [DFF99, HP00, CRF00]). All these query languages aim at only *exact matching* of query conditions. That is, answers are found when those XML documents match the given query condition exactly. However, this may not be always the case in XML model.

Consider the following motivating example in Figure 1.2 adopted from [GJK02]. All three XML data are similar in that all contain the information about the same `paper` published in `VLDB`. Furthermore, data D_1 and D_2 are more similar in their structures, while D_3 has a slightly different structure. It might be the case that D_1 and D_2 are originated from the same DTD while D_3 is an instance of a different DTD. Such case is often found in heterogeneous databases that integrates



(a) D_1 (b) D_2 (c) D_3

Figure 1.2: Three similar `paper` data in XML model.

XML data from diverse sources. However, using the proposed XML query languages and their notion of the "exact matching", it is not straightforward to find out that all three data are in fact describing the same `paper` instance due to their subtle differences in both structures and values.

To address such difficulties, in this dissertation, we propose to develop a query relaxation framework for searching answers that match the given query conditions *approximately*. *Query relaxation* enables systems to relax the user query to a *less* restricted form to permit approximate answers. Query relaxation technique has been used in relational databases (e.g., [MSB98, CCH94, CYC96, CG99, Gaa97]) and has proven to be a valuable technique for deriving approximate answers.

In XML domain, the need for such query relaxation increases since the flexible nature of XML model allows varied structures and/or values and the non-rigid XML tag syntax enables to embed a wealth of meta information in XML documents. The following points illustrates that query relaxation becomes more important for XML model than relational model:

1. Unlike in relational model where users are given a relatively small-sized schema to ask queries, the schema in XML model is substantially bigger and more complex. Therefore, it is often unrealistic for users to understand the full schema and compose very complex queries at once. In such a scenario, it becomes critical to be able to relax the user's query when the original query yields null or not sufficient answers.

2. As the number of data sources available on the web increases, it becomes common to build systems where data are drawn from heterogeneous data sources, where the structures of the participating data sources are different, although they employ the same ontologies with the same contents. Therefore, the capability to query against differently-structured data sources becomes more important [KNS99, KS01]. In such a setting, query relaxation technique allows query to be structurally relaxed and routed to diverse data sources with different structures.

3. Increased number of textual documents are converted to XML format for ease in information exchange and web presentation. To provide content-based retrieval of these documents, approximate matching of content becomes essential in order to avoid null answers [CJK00].

Throughout this dissertation, we explore diverse issues needed to support query relaxation for XML model.

1.2 Research Problems

The introduction of query relaxation for XML model brings up a myriad of challenging technical issues as follows:

- **Formal Framework**: The notion of query relaxation in XML model exhibits several novel relaxation characteristics because of the hierarchical nature of XML model. To precisely understand and describe such relaxation process in XML model, a well-defined framework that is capable of expressing most meaningful relaxation requirements in a precise and declarative manner is desired. How to devise such a framework that is both easy to understand and powerful to express is one interesting research area that we study.

  Such framework can be used as a basis to design and implement a query language that supports relaxation features in future. Furthermore, methods for efficiently evaluating relaxed XML queries can also be devised in the context of such framework.

- **Distance between Trees**: One of the remaining questions to support query relaxation is how to *rank* approximate answers according to their similarity to the original query conditions. Some of the invariants that we may think of are:

  - Exactly-matched answers should rank higher than approximate answers.
  - For three distinct answers A, B, and C, If A is ranked higher than B, and in turn B is ranked higher than C, then A must be ranked higher than C (i.e., transitivity)

It is relatively straightforward to devise a distance metric in relational model where each answer is a tuple with a set of attributes. In XML model, however, it is not clear what the notion of "distance" should be while retaining the above invariants for the tree-shaped XML data. For instance, in Figure 1.2, one cannot bluntly conclude that (c) be closer to (a) than to (b). In general, devising the right notion of distance (or similarity) between tree-shaped objects is a challenging task due to its inherent subjective nature. Therefore, our goal is to investigate plausible distance metrics useful for XML model and query relaxation context.

- **Selectivity Estimation**: The general form of query relaxation paradigm works as follows. When a given query Q is not *satisfactory* by users, system starts relaxation.

```
1. While (not Satisfactory($Q$))
    (a) $Q \leftarrow$ Relax($Q$)
    (b) Evaluate($Q$)
```

  The satisfactory condition varies depending on the users and applications. One of the popular satisfactory conditions is known as *at-least* k condition where users request "at least" k number of answers to be returned. This is a typical query pattern, for instance, when users try to locate a certain number of apartments meeting certain conditions. In such a setting, if the initial answer set does not satisfy the threshold k, then system starts query relaxation. A modified control flow for the "at-least k" scenario is as follows:

```
1. Answer $\leftarrow$ **Evaluate**($Q$)
2. While ($|$Answer$| < k$)
```

```
    (a) $Q \leftarrow$ Relax($Q$)
    (b) Answer $\leftarrow$ **Evaluate**($Q$)
```

The main problem of such execution flow lies in the expensive cost in executing the **Evaluate()** function inside of the loop many times. The plausible solution to such problem is to *estimate* the selectivity of the query, instead of *evaluating* the query, in checking the condition. Since estimating query selectivity typically costs must cheaper than evaluating the query, the gain in terms of computational cost is substantial. Assuming the existence of the function **EstimateSelectivity()**, the control flow can be simplified as follows:

```
1. $N \leftarrow$ **EstimateSelectivity**($Q$)
2. While ($N < k$)
    (a) $Q \leftarrow$ Relax($Q$)
    (b) $N \leftarrow$ **EstimateSelectivity**($Q$)
```

Therefore, techniques for accurately estimating selectivity for relaxed queries play an important role in query relaxation framework and thus part of the focus of our study. We especially study the problem to find a total selectivity **EstimateSelectivity**($R_1 + ... + R_n$) when a set of relaxed queries R_1, ..., R_n are generated from an original query Q.

- **Data Conversion**: One can envision a whole spectrum of ways to support XML query relaxation in database systems. In one end of the spectrum, one may build a native XML database system from the scratch that are capable of understanding and processing XML queries and their relaxation features. In the other end of the spectrum, instead of modifying database engines, one may convert XML data into known data formats and reuse conventional database systems directly. For instance, if all the information in original XML data can be correctly converted to relational tuples in a lossless fashion, then query relaxation can be supported via relational database systems (after XML queries are properly converted to SQL).

Therefore, being able to convert data between XML and relational formats efficiently and effectively plays an important role in supporting query relaxation for XML model. However, transforming a hierarchical XML model to a flat relational model and vice versa is not a trivial task. There are several difficulties issues as follows [STH99]:

- Non 1-to-1 mapping
- Set values
- Recursion
- Fragmentation

Therefore, our goal in this study is to devise XML-to-Relational and Relational-to-XML conversion methods that captures the original information as correctly as possible in the final results. Especially, we focus on the issues of capturing semantic constraints of the original data.

1.3 Outline of the Dissertation

The dissertation is organized as follows.

- **Chapter 2** gives a brief overview on XML model and a detailed comparative analysis of the XML schema and query languages.
- **Chapter 3** discusses on the formal framework to precisely capture and describe query relaxation process of XML model.

- Once the framework has been laid out, we explore various technical issues that are needed to support query relaxation for XML model. First, a new notion of distance (or similarity) among XML data is discussed in **Chapter 4**.

- Second, an extended estimation technique for query selectivity is discussed in **Chapter 5**.

- Using mature relational databases is one feasible approach so as to support query relaxation for XML model. Towards this end, in **Chapters 6** and **7**, three conversion algorithms between XML and relational models are introduced.

- Finally, we conclude our work in **Chapter 8** with discussion about the remaining problems in query relaxation in general and our work plan of the future.

CHAPTER 2

The XML Model

The XML (eXtensible Markup Language) [BPS00] is the new universal format for structured documents and data on the Web, currently being standardized by the World Wide Web Consortium. XML is designed to improve the functionality of the Web by providing more flexible and adaptable information identification. It is called *extensible* because it is not a fixed format like HTML (HyperText Markup Language) [RHJ99], which is a pre-defined markup language, primarily for displaying data on Web. Instead, XML is rather a *meta-language* that can be used for defining other languages. That is, with XML as a tool, one can define his/her own customized markup languages for specific types of documents. In other words, while HTML is one type of a bread, XML can be viewed as a bread maker to bake various kinds of breads. XML is a subset of simple conventions from SGML (Standard Generalized Markup Language), the international standard meta-language for text markup systems (ISO 8879), without some of the more esoteric features of SGML. In this Chapter, we briefly present an overview of XML model and its schema and query languages [BL01].

2.1 The Basics

An XML document typically consists of two major components – *schema* and *data*. The *schema* is for describing the *data* and is specified in one of the proposed schema language notations. One of the most popular XML schema languages is DTD (Document Type Definition) [BPS00].

XML is a textual representation of the hierarchical data model. The meaningful piece of the XML document is bounded by matching starting and ending *tags* such as `<name>` and `</name>`. The main building blocks of XML model are *element* and *attribute* as follows:

```
<elem-name attr-name='attr-val'> elem-content </elem-name>
```

In DTD, elements and attributes are defined by the keywords `<!ELEMENT>` and `<!ATTLIST>`, respectively. In general, components in DTD are specified by the following BNF syntax:

```
<!ELEMENT> <elem-name> <elem-content-model>
<!ATTLIST> <attr-name> <attr-type> <attr-option>
```

Element content model is the logical structure of the element contents based on the regular expressions such as "?" (0 or 1 instance), "*" (0 or many instances), or "+" (1 or many instances). For instance, in the following example, the element `paper` contains only one instance of sub-element `title`, one or many instances of sub-element `author`, and zero or many instances of sub-element `citation`:

```
<!ELEMENT paper    (title,author+,citation*)>
```

2.2 XML Schema Languages

Since the requirements of the application vary, one may choose different XML schema language to describe his/her XML data. Among a dozen XML schema languages recently proposed, we briefly review the six representative schema languages and compare their features [LC00a].

1. XML DTD (DTD in short), a subset of SGML DTD, is the *de facto* standard XML schema language of the past and present and is most likely to thrive until XML-Schema finally arrives. It has limited capabilities, compared to other schema languages. Its main building block consists of an *element* and an *attribute*. The real world is typically represented by the use of hierarchical element structures. The 2nd Edition of the language specification became a W3C Recommendation on October 6, 2000.

2. According to [MM99], XML-Schema language is intended to be more expressive than DTD while it can be expressed in XML notations and more usable by a wider variety of applications. It has many novel mechanisms such as inheritance for attributes and elements, user-defined datatypes, etc. XML-Schema became W3C Recommendation on May 2, 2001. The specification is stable and has been reviewed by the W3C Members, who favors its adoption by academic, industry, and research communities.

3. SOX (Schema for Object-Oriented XML) is an alternative schema language for defining the syntactic structure and partial semantics of XML document types. SOX leverages on the object-oriented relationships between data structures to allow the easy management of a component library, and the use of type relationships within schema-based e-commerce applications. The current version, 2.0, is developed by Commerce One. The xCBL (XML Common Business Library) 3.0 is implemented using SOX version 2.0.

4. Schematron, created by Rick Jelliffe, is quite unique among the others, since it focuses on *validating* schemas using patterns instead of *defining* schemas. Its schema definition is simple enough to be defined in a single page, and it still provides powerful constraint specification via XPath [CD99]. The latest version is 1.5.

5. DSD 1.0 was co-developed by AT&T Labs and BRICS with the goals of context-dependent description of elements and attributes, flexible default insertion mechanisms, expressive power close to XSLT [Cla00], etc. Like Schematron, DSD puts a strong emphasis on *schema constraints*. DSD has been used in an AT&T application, named XPML. A revised specification, DSD 1.1, is under development.

6. RELAX (REgular LAnguage description for XML) is a novel XML schema language developed by Makoto Murata. RELAX has been standardized by JSA (Japanese Standard Association) and was submitted to ISO (International Standard Organization) as a fast track procedure document for DTR processing and recently has been approved by the ISO/CS ballot [ISO00]. RELAX is based on clean principles of hedge automata [Mur00a] and has the capability of expressing context-sensitive schema rules by means of non-terminal symbols. At the time of writing, RELAX and TREX [Cla01] are being merged into RELAX-NG [CM01] under OASIS Technical Committee.

2.2.1 Comparison

XML schema languages can be classified as shown in Figure 2.1. Clearly, some has constraints-oriented features while others have structure-oriented features. Furthermore, the six XML schema languages can be organized into the following three classes based on their expressive powers, as depicted in Figure 2.2.

1. **Class 1**: DTD is a representative of **Class 1 of core schema language for XML**, with a restricted expressive power. It does minimally support the basic schema abstractions and severely lacks datatypes and constraints definition. Since the expressive power of DTD is strictly weaker than other

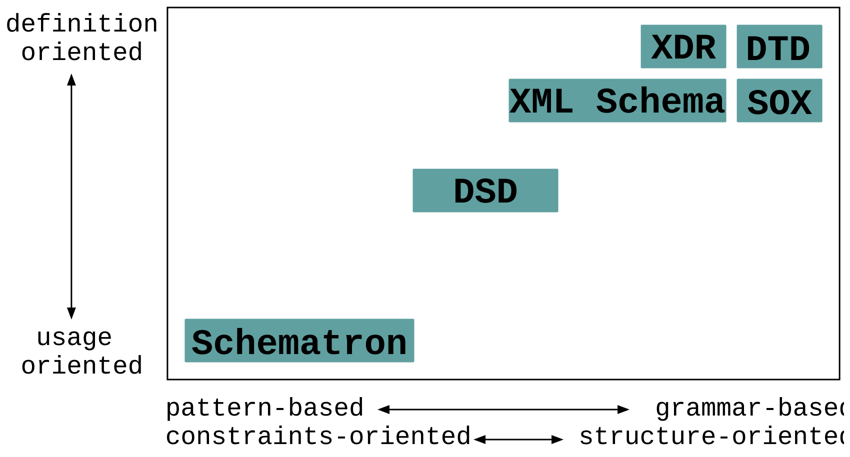


Figure 2.1: Feature-based classification of XML schema languages.

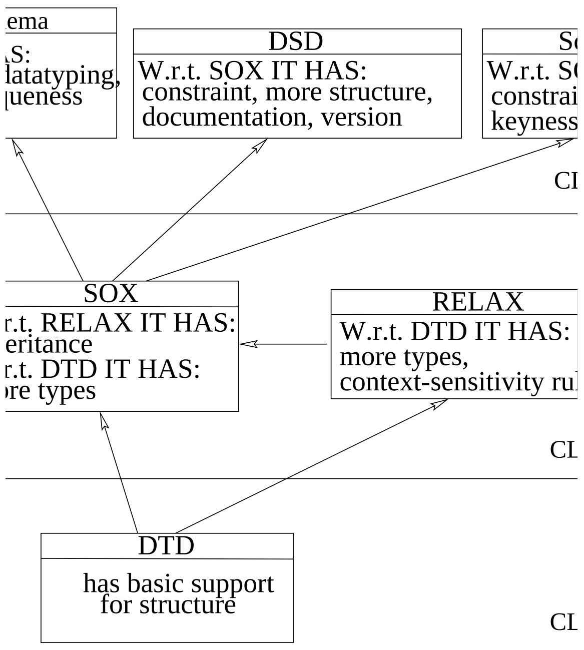


Figure 2.2: Expressive power-based classification of XML schema languages.

schema languages, the translation from other schema languages to DTD is straightforward while the reverse translation is likely to be lossy.

2. **Class 2**: Both RELAX and SOX belong to the middle tier and can be considered as representatives of **Class 2 of extended schema language for XML**. Their support for datatypes is not enough (e.g., lack of explicit null and user-defined type) although basic schema abstractions can be supported rather sufficiently. In addition, they lack the support for the full range of content models and database-oriented features such as uniqueness and keyness. Like DTD they mostly fail to fully handle constraint specifications.

3. **Class 3**: XML-Schema, Schematron and DSD are the most expressive languages and are representatives of **Class 3 of expressive schema languages for XML**. Whereas XML-Schema fully supports features for schema datatype and structure, Schematron provides a flexible pattern language that can describe the detailed semantics of the schema. DSD tries to support common features supported by XML-Schema (e.g, structure) and Schematron (e.g., constraint) along with some additional features.

2.3 XML Query Languages

As the amount of XML data on the Web increases, the ability to access and query the data becomes increasingly important. Towards this goal, several XML query languages have been proposed. Similar to XML schema languages, let us briefly review the six representative query languages and compare their features [BC00].

1. Lorel[1] was originally designed for querying semistructured data [AGM97] and has now been extended to XML data [AGM97]; it was conceived and implemented at Stanford University. It is a user-friendly language in the SQL/OQL style, it includes a strong mechanism for type coercion and permits powerful path expressions, useful when the structure of a document is not known in advance [AQM97].

2. XML-QL[2] was designed at AT&T Labs; it has been developed as part of Strudel Project. XML-QL language extends SQL with an explicit `CONSTRUCT` clause for building the document resulting from the query and uses the *element patterns* (patterns built on top of XML syntax) to match data in an XML document. XML-QL can express queries as well as transformations, for integrating XML data from different sources [DFF99].

3. XML-GL is a graphical query language, relying on a graphical representation of XML documents and DTDs by means of labeled *XML graphs*. It was designed at Politecnico di Milano; an implementation is ongoing and a web site is under construction[3]. All the elements of XML-GL are displayed visually; therefore, XML-GL is suitable for supporting a user-friendly interface (similar to QBE) [CCD99]. The last specification of the language can be found in [CDF01].

4. The Extensible Stylesheet Language (XSL) has facilities that could serve as a basis for an XML query language, XSLT (Extensible Stylesheet Language Transformations). An XSLT stylesheet consists of a collection of template rules; each template rule has two parts: a pattern which is matched against nodes in the source tree and a template which is instantiated to form part

[1]http://www-db.stanford.edu/lore
[2]http://www.research.att.com/sw/tools/xmlql
[3]http://xerox.polimi.it/Xml-gl

of the result tree. XSLT makes use of the expression language defined by XPath [CD99] for selecting elements for processing, for conditional processing and for generating text. It was designed by the W3C XSLT working group [Cla00, SLR98, Gro98]. XSLT is still under development at the time of writing. XSLT 1.1 is a working draft [Cla00]. Future developments are described in the requirements specification of XSLT 2.0 [MS01].

5. XQL is a notation for selecting and extracting XML data. XQL can be considered a natural extension to the XSLT pattern syntax; it is designed with the goal of being syntactically simple and compact (a query could be part of a URL), with a reduced expressive power [RLS98, Rob99, SLR98]. The language has been reviewed in 1999 [Rob99].

6. XQuery is the first W3C proposal for a standard query language, published in February 2001 and revised in June 2001 [CFR01]. The current proposal of XQuery version 1.0 is mostly drawn from Quilt [CRF00], a newly-conceived query language for XML, which inherits the experiences of several past query languages and attempts to unify them. XQuery assembles many features from previously defined languages, such as the syntax for navigation in hierarchical documents from XPath and XQL, the notion of binding variables from XML-QL, the combination of clauses a-la SQL and the notion of a functional language from OQL; it is designed with the goal of being expressive, of exploiting the full versatility of XML and of combining information from diverse data sources [CFR01].

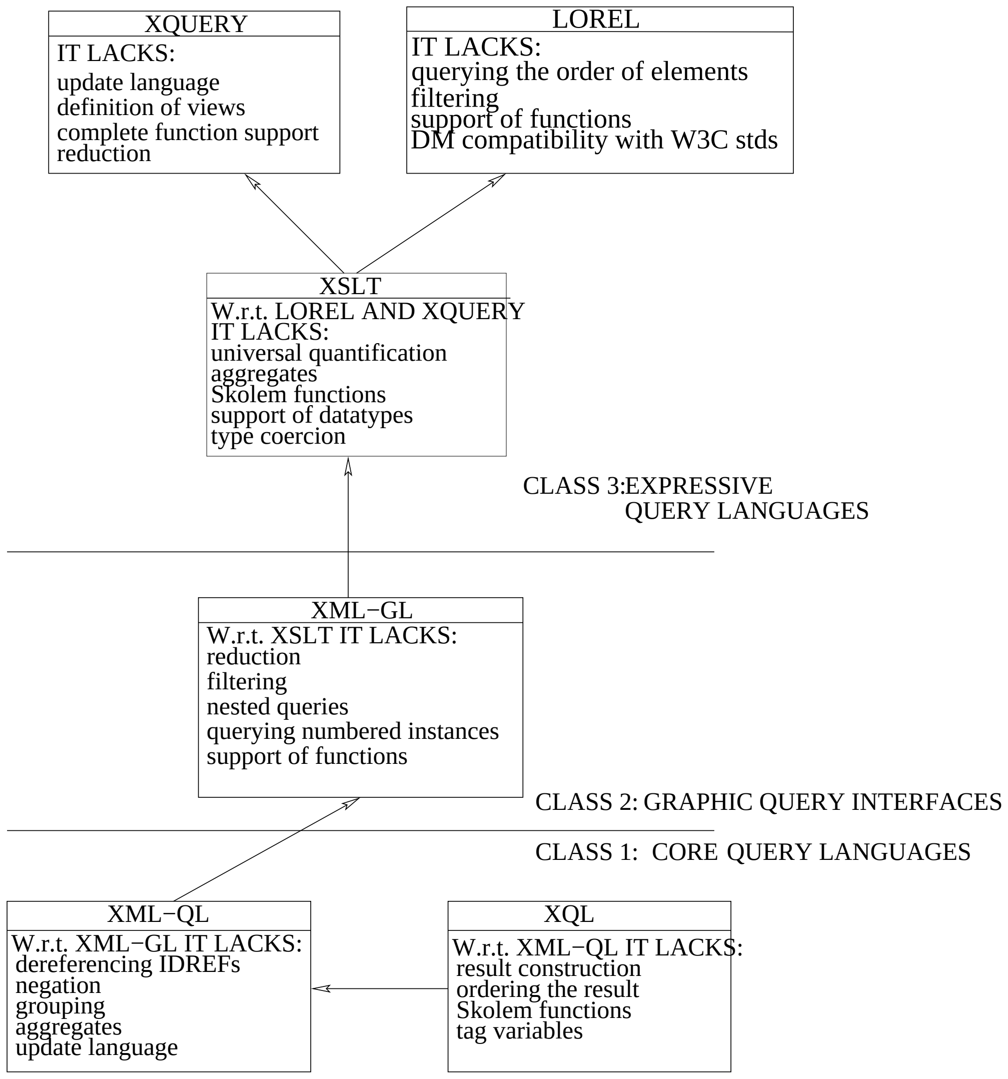


Figure 2.3: Expressive power-based classification of XML query languages.

2.3.1 Comparison

The six XML query languages can be organized into the following three classes as depicted in Figure 2.3.

1. **Class 1**: The new XQL and XML-QL are representatives of **Class 1 of core query languages for XML**, playing the same role as core SQL standards and languages (e.g., the SQL supported by ODBC) in the relational world. Their expressive power is included within the expressive power of XSLT.

2. **Class 2**: XML-GL can be considered a representative of **Class 2 of graphical query interfaces to XML**, playing the same role as graphical query interfaces (e.g., QBE) in the relational world. The queries being supported by XML-GL are the most relevant queries supported by XSLT. It can suitably be adopted as a front-end to any of these query languages (more or less powerful), to express a comprehensive class of queries (a subset of them in case of more powerful languages).

3. **Class 3**: XQuery, Lorel and XSLT are representatives of **Class 3 of expressive query languages for XML**, playing the same role as high-level SQL standards and languages (e.g., SQL2) in the relational world. XQuery and Lorel are quite different in their syntax and semantics, due to their completely distinct nature (the semi-structured approach of Lorel as opposed to the XML-inspired mainstream of XQuery). Moreover, Lorel is strongly object-oriented, while XQuery can be considered value-oriented. XQuery is a promising expressive query language, that realizes its potentiality by incorporating the experience of XPath and XQL on one side, of SQL, OQL and XML-QL on the other side. The third language of this class, XSLT,

covers a lower position in the taxonomy being less powerful than the previous two. It is a stylesheet language with a fairly procedural tendency, as opposed to Lorel, which can be considered completely declarative, and to XQuery, which blends a declarative and procedural flavor.

2.4 Summary

As more data are stored natively in XML format or converted to XML format from legacy formats, the importance of understanding XML data model from the perspective of database becomes evident. In fact, the arrival of XML model brings abundant interesting research opportunities to the conventional database research: XML storage system, XML data conversion, XML data indexing, XML query language, XML query evaluation, etc. To be better positioned to cope with such challenging issues, in this Chapter, we presented an overview of XML model and distinctive features of various XML schema/query language proposals.

CHAPTER 3

XML Relaxation Framework

Query relaxation enables systems to weaken the given query constraints to a less restricted form to accommodate user's needs and has been extensively investigated and used in both IR and DB areas. With the arrival of XML model, its importance becomes even more evident; due to the heterogeneity of XML data, it is often more useful to permit approximate matching of XML queries than to return only exact answers. In this Chapter, we discuss some of the basic technical issues that have emerged in an initial exploration of the topic.

3.1 Background

Traditionally, queries submitted by users are *modified* in various aspects and ways to cope with different situations. The importance of such techniques that enable automatic query modification stems from the fact that this behavior is very common activity in human discourse. For instance, if a customer asks a travel agent asking the flight from the city X to the city Y on the date of Z, then when no satisfactory flights are found, the agent is very likely to respond with alternatives by "changing" one or many of X, Y, and Z of the initial query [GGM90]. In general, *query modification* broadly describes the process of changing a query when the answer to the query does not meet the expectations of the user. In the relational database jargon, this "change" to a query can occur in two different places – condition (i.e., FROM and WHERE clauses) and projection (i.e., SELECT clause) parts. Intuitively, changing the condition part aims to find some meaningful answers related to what the user specified, while changing the projection part aims to find additional aspects of answers than the user specified. In the travel agent example, for instance, changing condition part corresponds to the case of changing any of X, Y, or Z in the query. On the other hand, changing projection part may correspond to the case of finding hotel and weather information in addition to the flight schedule.

In XML domain, the need for such query relaxation increases since the flexible nature of XML model allows varied structures and/or values and the non-rigid XML tag syntax enables to embed a wealth of meta information in XML documents. There are several important differences that make the query relaxation for XML model more important than that for the relational model, as discussed in Chapter 1.1.

3.2 Related Work

Query relaxation and its related techniques (e.g., cooperative information systems, query expansion, etc.) have been extensively investigated in both IR and DB areas (e.g., [Kap82, Mot84, Mot86, Gal88, Mot90, CLC91, CYC94, CCH94, CYC96, CCH96, God97, CG99]). In this Section, we briefly review those related works.

When a query fails, it is more *cooperative* to identify the causes of failure, rather than just to report the empty answer set [God97]. Information system with such capability is known as *Cooperative Information System*. Kaplan [Kap82] is probably the first to observe the relevance of false "presuppositions" to databases and studied a method to find the minimal failing sub-queries. Also, he introduced the notion of generalizing a failing query into a successful query by removing some of the failing sub-queries from the original query.

Motro [Mot84, Mot86, Mot90] extended the Kaplan's notion of query generalization into the case where a degree of query condition is *relaxed*. Thus, Kaplan's query generalization where some sub-queries may be removed can be viewed as a special case of Motro's extended framework.

CoBase system [CLC91, CCH94, CYC96, CCH96] supports query generalization (also known as query relaxation) for relational data model by automatically mining conceptual hierarchies (i.e., TAH) of predicates and terms over data. Their query relaxation technique has also been applied to medical image search via approximate feature (e.g., size, location) and contents matching [CHC98], logistic planning application for searching objects with similar characteristics in the neighboring desired locations [CYC94], etc.

Cooperativeness of the information system with the focus on the integrity constraints was studied in [Gal88]. In [GGM90, Gaa97], Gaasterland et al introduced query relaxation as a platform of cooperative answering from the perspective of deductive database and logic programming. Godfrey [God97] presents the theoretical complexity results of *minimal failing sub-query* and *maximal succeeding sub-query* problems, which are important issues in, for instance, relaxing the failing query to the successful one with smallest gap in answer space.

In terms of the formal framework, Chaudhuri [Cha90] proposed an elegant one to describe query modification, and especially query generalization, for the relational model. In this Chapter, using his framework, (1) we investigate the types and semantics of relaxations in the XML model, (2) by converting XML model appropriately, we show that Chaudhuri's model can capture the variety of query relaxations for XML model, and (3) by extending Chaudhuri's model slightly, we present a new framework, termed as $\mathcal{Q}AC$, where one can control the way query relaxation is processed in further details.

In IR field, much research can be found on *Query Expansion* which is the technique to (semi-)automatically add (thus *expansion*) related keywords to the original user query (i.e., keyword list) to yield better precision and recall [BR99]. Techniques typically consider either lexical similarities between keywords (e.g., "phen$\underline{e}$menon" is changed to "phen$\underline{o}$menon") or human-built structures such as WordNet [Fel98] to find similar concepts (e.g., "surname" is a synonym of "familyname"). Although query expansion in IR is related to query relaxation in this dissertation, the focus of query relaxation for XML model is substantially different. That is, unlike expansion in IR which focuses on "which" keywords to add, we focus on relaxing both structures (i.e., edge relationship) and values (i.e., contents). Furthermore, we investigate related techniques such as distance metric or selectivity estimation to support query relaxation for XML model.

From Web perspective, several authors have proposed a language/system that supports approximate pattern matching and answer ranking. In [TW00], a similarity operator "$\sim$" for XML is presented. In their work, "$\sim$" measures the similarity between the value of element/attribute and the given constant helped by underlying thesaurus. In [FG01], authors describe XIRQL, an extension of XQL [RLS98], that integrates IR features by supporting weighting and ranking, relevance-oriented search, and data types with vague predicates.

More recently, query relaxation techniques relevant to XML model start to appear. For instance, [ACS02] proposes three relaxation schemes – generalizing nodes, deleting nodes, and relaxing edges – and investigates their evaluation issues. Also, [KS01] proposes another kind of relaxation based on the order of nodes and presents various complexity results on query evaluation. However, neither of [ACS02, KS01] deal with the issues such as distance metric or selectivity estimation for relaxed queries. In this Chapter, we consider three relaxation primitives for XML model proposed in [ACS02].

3.3 A Framework for XML Query Relaxation

Let us precisely define what the query modification and query relaxation are. First, we review some terminologies. We borrow the following notations from [Cha90]. A *query* is an an *open formula*, denoted $Q(\vec{x})$, where $\vec{x}$ is the set of *free variables*. For the simplicity of exposition, in this Chapter, we will restrict ourselves to only *conjunctive queries*, $Q(\vec{x}) \equiv \exists \vec{z} \bigwedge_i P_i(\vec{x_i})$ where $\vec{x_i} \subseteq (\vec{x} \cup \vec{z})$ and P_i-s are relation symbols. The answer relation represented by $Q(\vec{x})$ against a database D is denoted by $\langle Q(D) \rangle$, or simply $\langle Q \rangle$ when context is clear. Set relationships between two answer relations, $\langle Q \rangle$ and $\langle R \rangle$, can be straightforwardly defined. Then, the query modification is formally defined as follows:

Definition 1 (Query Modification) *A query Q is said* modified *to R if the following holds:*

$$\langle Q \rangle \cap \langle R \rangle \neq \emptyset \qquad \Box$$

That is, we are only interested in the query modification as long as the query is changed such that the modified query still shares some common answers with the original query. Now, when a query Q is modified to a query R, the change can be categorized into one of the four notions: (1) *Query Rewrite*: Q is re-written to R that generates the same set of answers with different characteristics such as faster computation (e.g., query rewrite in the semantic query optimization problem [JK84]), (2) *Query Restriction*: the scope of Q is restricted so that

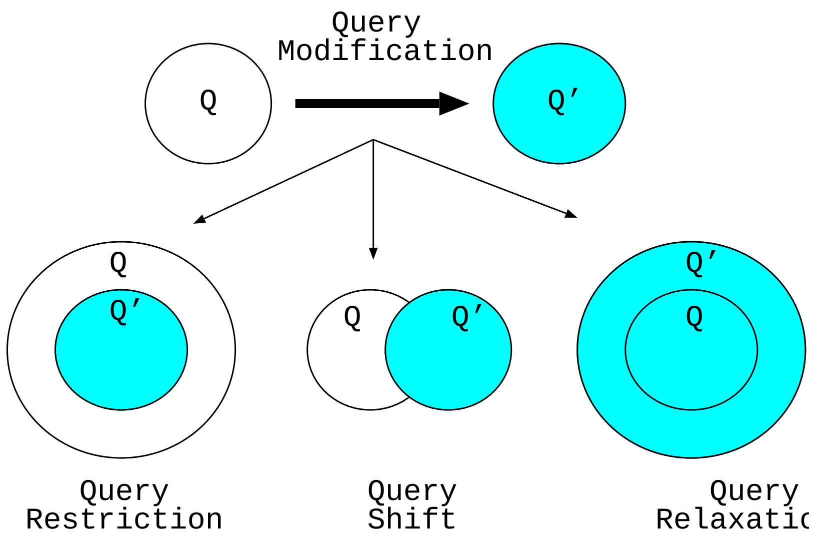


Figure 3.1: Illustration of query modifications.

less number of or smaller-scoped answers will be returned (e.g., top-k selection problem [CG99]), (3) *Query Relaxation*: the scope of Q is relaxed so that more number of or bigger-scoped answers will be returned, and (4) *Query Shift*: the scope of Q is shifted to generate partially-overlapping, but different set of answers. Their corresponding properties are as follows:

$$\text{Query Rewrite} \quad : \quad \langle Q \rangle \equiv \langle R \rangle \tag{3.1}$$

$$\text{Query Restriction} \quad : \quad \langle Q \rangle \supseteq \langle R \rangle \tag{3.2}$$

$$\text{Query Relaxation} \quad : \quad \langle Q \rangle \subseteq \langle R \rangle \tag{3.3}$$

$$\text{Query Shift} \quad : \quad \langle Q \rangle \not\supseteq \langle R \rangle \wedge \langle Q \rangle \not\subseteq \langle R \rangle \tag{3.4}$$

Our focus, in this Chapter, is especially the **Query Relaxation**. As the above property implies, query relaxation enables systems to relax the given query constraints to a less restricted form such that new answer set is the "*superset*" of the original answer set. Pictorial illustration of different types of query modification is given in Figure 3.1.

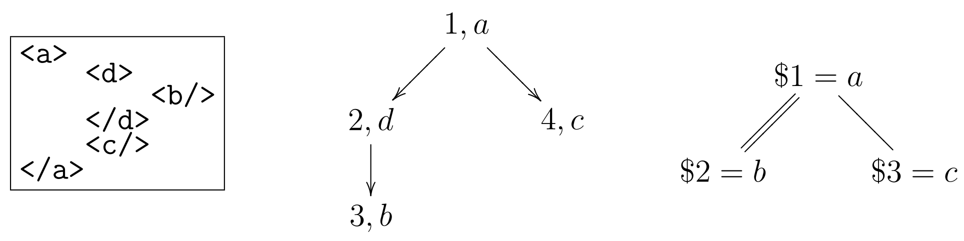
Chaudhuri's framework [Cha90] captures query modifications for the relational model. Due to the characteristics of XML model, however, new kinds of query relaxations appear in XML model. Also, Chaudhuri's model does not allow fine tunings of the relaxation process. To remedy these shortcomings, we propose a framework, termed as $\mathcal{Q}AC$, where one can precisely control the way query relaxation is processed.

Definition 2 ($\mathcal{Q}AC$ Framework) *A query S is represented by triple: $S \equiv (\mathcal{Q}, \mathcal{A}, \mathcal{C})$, where:*

- *$\mathcal{Q}$ is a (labeled) conjunctive relational query, $\bigwedge_i L_i : P_i$, where L_i is an optional unique label to identify the conjunctive term P_i.*
- *$\mathcal{A}$ is a boolean function, called* Acceptance Test, *that takes as input the answer generated by executing the query $\mathcal{Q}$ over a database D, and returns $\{$True, False$\}$.*
- *$\mathcal{C}$ is a statement, called* Control Statement, *that guides the relaxation process as the user specifies.* $\square$

The intuitive procedure of the query relaxation in $\mathcal{Q}AC$ framework is as follows: After the query $\mathcal{Q}$ is executed, its acceptance is tested using $\mathcal{A}$. If it is acceptable (i.e., $\mathcal{A}(\langle\mathcal{Q}\rangle) = \text{True}$), then the answer is returned to the user. Otherwise, $\mathcal{Q}$ is *relaxed* guided by $\mathcal{C}$ to a relaxed query $\mathcal{Q}'$.

XML document is, in this Chapter, represented as a tree, where each element (regardless of being tag, CDATA, etc.) becomes a node and an element and its sub-element (or attribute) relationship becomes an edge. To simplify discussion, let us assume that each node in a tree has two values: the *id* that uniquely



```
<a>
    <d>
        <b/>
    </d>
    <c/>
</a>
```

(a) An XML document (b) Tree representation (c) An XML query

Figure 3.2: XML query representation as a tree.

identifies the node and the v of the label of the node. Similarly, an XML query against such tree can be naturally represented as a tree, where each node has a form "variable = value" and each edge is a constraint of either parent-child or ancestor-descendent relationship, denoted as single or double edges, respectively. Each variable is denoted $\$i$, for integer i. More importantly, this XML query can be *equivalently* represented by using three relations – `node(id,v)`, `pc_edge(v,w)`, `ad_edge(v,w)` – in a conjunctive query notation. These notations are inspired by [JLS01]. We will use both notations interchangeably in the rest of the Chapter. An example is illustrated in Figure 3.2.

As an example of query relaxation in $\mathcal{QAC}$ framework, consider the following scenario. Suppose one wants to find at-least k answers for the given query in Figure 3.2(c), that finds all occurrences of the tag `<a>` that has a descendant `<b>` and a child `<c>`. Then, this can be captured as $S \equiv (\mathcal{Q}, \mathcal{A}, \epsilon)$, where

$$\begin{aligned} \mathcal{Q}(\$1,\$2,\$3) &\equiv node(\$1,a) \wedge node(\$2,b) \wedge node(\$3,c) \wedge \\ &\quad ad\_edge(\$1,\$2) \wedge pc\_edge(\$1,\$3) \\ \mathcal{A}(\langle\mathcal{Q}\rangle) &\equiv count(\langle\mathcal{Q}\rangle) \geq k \end{aligned}$$

After an evaluation of $\mathcal{Q}$ returns only 1 answer $(\$1,\$2,\$3) = (1,3,4)$ as in Figure 3.2(b), the query $\mathcal{Q}$ will be relaxed further to satisfy the acceptance test of at-least k.

In the subsequent Section, we will describe, using $\mathcal{QAC}$ framework, various forms of XML query relaxations; some relaxations appeared in the relational model as well while other relaxations are novel and due to the characteristics of XML model.

3.4 Types and Semantics of XML Query Relaxation

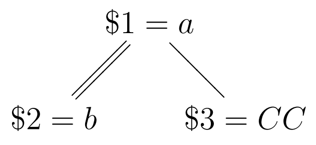
Query relaxation in the relational databases typically focuses on the "value" aspect (e.g., [CYC96]). For instance, a relational query "find persons with a salary range 50K – 55K", if there is no answer or not sufficient answer, may be relaxed to a query "find persons with a salary range 45K – 60K". In XML model, in addition to the value relaxation, new types of query relaxation, known as *structural relaxation* and *order relaxation*, are introduced as follows.

3.4.1 Value Relaxation

In XML context, *value relaxation* is expanding the value scope of certain nodes to allow matching additional answers. A value can be relaxed to a range of numeric values or a set of non-numeric values. In general, such relationship can be induced by the natural hierarchies or partial order among domain values [Cha90]. For instance, if `person` is a super-type of types `student`, `TA`, or `RA`, then a partial order "{`student`, `TA`, `RA`} ≤ `person`" exists. Then, with the following query rewrite rule on the conjunctive term P,

$$\forall \vec{x} \forall a \forall b ((a \leq b) \rightarrow (P(\vec{x}, a) \rightarrow P(\vec{x}, b))$$

a value "`student`" can be relaxed to "`person`", allowing the relaxed query matching. Figure 3.3 illustrates a value relaxation applied to the query of Figure 3.2(c), where "CC" is a supertype of "C". After the value relaxation, the new relaxed



$\equiv$ node($1,a) $\wedge$ node($2,b) $\wedge$ node($3,CC) $\wedge$ ad_edge($1,$2) $\wedge$ pc_edge($1,$3)

Figure 3.3: Example of a relaxed query with value relaxation.

query of Figure 3.3 may match additional answers, e.g., `<a><b/><CC/></a>`. The query relaxation developed in [CYC96, CG99] can be extended to support the value relaxation in XML model.

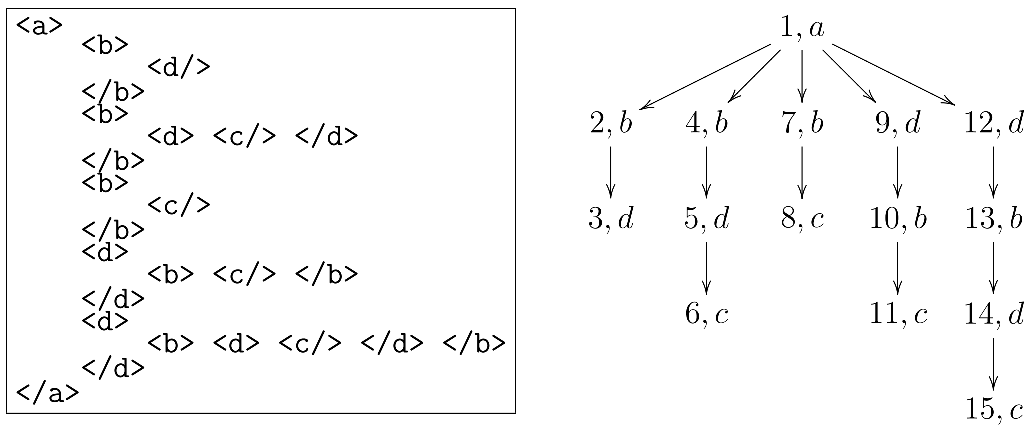
3.4.2 Edge Relaxation

In this relaxation type, a parent-child edge can be relaxed to an ancestor-descendent edge. That is, the semantics of the relaxation is that while the original query finds answers with only a parent-child relationship, the new query will be able to find answers with an ancestor-descendent relationship which is, by definition, a superset of a parent-child relationship. Consider a simple path expression $/a/b/c$ in XPath notation (i.e., find answer that has a as a root that in turn has b as a child that in turn has c as a child). This can be captured in $\mathcal{QAC}$ framework as follows: $S \equiv (\mathcal{Q}, \epsilon, \epsilon)$, where

$$\begin{aligned}\mathcal{Q}(\$1, \$2, \$3) \;\equiv\; & node(\$1, a) \wedge node(\$2, b) \wedge node(\$3, c) \wedge \\ & pc\_edge(\$1, \$2) \wedge pc\_edge(\$2, \$3)\end{aligned}$$

Then, the possible edge relaxations that can occur are, in XPath notation, $r_1 : /a//b/c$, $r_2 : /a/b//c$, and $r_3 : /a//b//c$ and can be captured as follows:

$$\begin{aligned}R_1(\$1, \$2, \$3) \;\equiv\; & node(\$1, a) \wedge node(\$2, b) \wedge node(\$3, c) \wedge \\ & ad\_edge(\$1, \$2) \wedge pc\_edge(\$2, \$3) \\ R_2(\$1, \$2, \$3) \;\equiv\; & node(\$1, a) \wedge node(\$2, b) \wedge node(\$3, c) \wedge \\ & pc\_edge(\$1, \$2) \wedge ad\_edge(\$2, \$3)\end{aligned}$$



```
<a>
    <b>
        <d/>
    </b>
    <b>
        <d> <c/> </d>
    </b>
    <b>
        <c/>
    </b>
    <d>
        <b> <c/> </b>
    </d>
    <d>
        <b> <d> <c/> </d> </b>
    </d>
</a>
```

(a) An XML document

$$\begin{tikzcd}
 & & 1,a \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
2,b \arrow[d] & 4,b \arrow[d] & 7,b \arrow[d] & 9,d \arrow[d] & 12,d \arrow[d] \\
3,d & 5,d \arrow[d] & 8,c & 10,b \arrow[d] & 13,b \arrow[d] \\
 & 6,c & & 11,c & 14,d \arrow[d] \\
 & & & & 15,c
\end{tikzcd}$$

(b) Tree representation

Figure 3.4: Example of an XML data and its tree representation.

Q	\$1	\$2	\$3
	1	7	8

R_1	\$1	\$2	\$3
	1	7	8
	1	10	11

R_2	\$1	\$2	\$3
	1	7	8
	1	4	6

R_3	\$1	\$2	\$3
	1	7	8
	1	10	11
	1	4	6
	1	13	15

Table 3.1: Answers to the queries Q, R_1, R_2, and R_3 from XML data in Figure 3.4.

$$\begin{aligned} R_3(\$1,\$2,\$3) \;\equiv\; & \mathit{node}(\$1,a) \wedge \mathit{node}(\$2,b) \wedge \mathit{node}(\$3,c) \wedge \\ & \mathit{ad\_edge}(\$1,\$2) \wedge \mathit{ad\_edge}(\$2,\$3) \end{aligned}$$

The three new queries yield additional answers than the original query Q. For instance, consider an XML document in Figure 3.4. The *id* values of the nodes in (b) are assigned in pre-ordering. The different set of answers for the nodes \$1, \$2, and \$3 for queries Q, R_1, R_2, and R_3 are shown in Table 3.1, where the numbers represent the *id* of matching nodes. Clearly, R_1, R_2 and R_3 contain all the answers of Q, and also include their corresponding approximate answers.

Another type of edge relaxation, also known as *subtree promotion*, was introduced in [ACS02]. This relaxation permits a query subtree to be promoted so that the subtree is directly connected to its former grandparent by an ancestor-descendent edge. Therefore, in $\mathcal{QAC}$ framework, this will result in replacing "$pc\_edge(a, b) \wedge pc\_edge(b, c)$" with "$pc\_edge(a, b) \wedge ad\_edge(a, c)$". For instance, the relaxed query R_4 with subtree promotion from $\mathcal{Q}$ is the following:

$$\begin{aligned} R_4(\$1, \$2, \$3) \ \equiv \ & node(\$1, a) \wedge node(\$2, b) \wedge node(\$3, c) \wedge \\ & pc\_edge(\$1, \$2) \wedge ad\_edge(\$1, \$3) \end{aligned}$$

3.4.3 Node Relaxation

In this type of relaxation, certain constraints related to nodes in a query tree can be relaxed in several ways: (1) value attached to the node n can be "don't care" (i.e., "_"), permitting to match any non-null values. In $\mathcal{QAC}$ framework, then, all the conjuncts involving the variable n will have the form: $node(\$n, \_)$. (2) the node n can be removed while ensuring the "superset" property. That is, if n is a leaf node, it can be simply removed. However, if n is an internal node, the children of n nodes become grand-children of n [ACS02]. This case corresponds to the change to the projection part of the query and is equivalent to removing all the conjuncts, in $\mathcal{QAC}$ framework, involving the variable n accordingly.

Consider the simple path expression $Q : /a/b/c$ again. A few possible node relaxations are $S_1 : /a/b/\_$ (i.e., node c becomes don't care), $S_2 : /a/b$ (i.e., node c is removed) and $S_3 : /a//c$ (i.e., node b is removed). These relaxations can be captured as follows:

$$\begin{aligned} S_1(\$1, \$2, \$3) \ \equiv \ & node(\$1, a) \wedge node(\$2, b) \wedge node(\$3, \_) \wedge \\ & pc\_edge(\$1, \$2) \wedge pc\_edge(\$2, \$3) \\ S_2(\$1, \$2) \ \equiv \ & node(\$1, a) \wedge node(\$2, b) \wedge pc\_edge(\$1, \$2) \end{aligned}$$

Q	\$1	\$2	\$3
	1	7	8

S_1	\$1	\$2	\$3
	1	2	ϵ
	1	4	ϵ
	1	7	8

S_2	\$1	\$2
	1	2
	1	4
	1	7

S_3	\$1	\$3
	1	6
	1	8
	1	11
	1	15

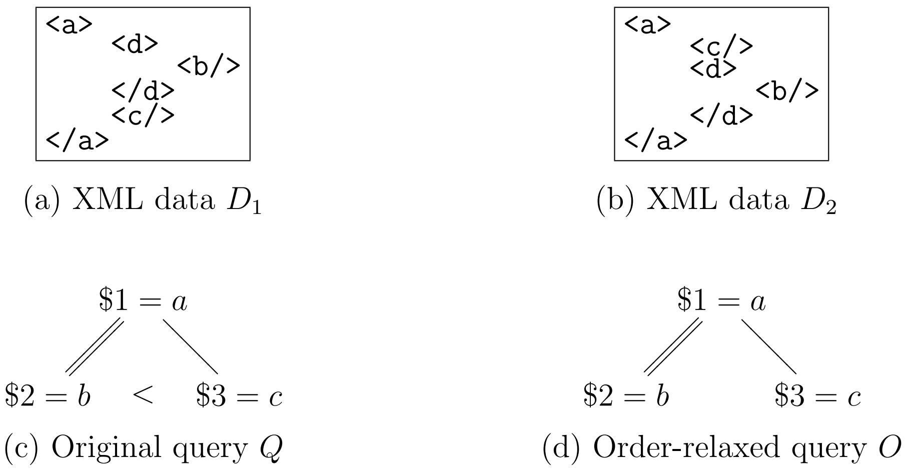
Table 3.2: Answers to the queries Q, S_1, S_2, and S_3 from XML data in Figure 3.4.

$$S_3(\$1, \$3) \quad \equiv \quad node(\$1, a) \wedge node(\$3, c) \wedge ad\_edge(\$1, \$3)$$

Table 3.2 illustrates a set of matching answers of queries Q, S_1, S_2, and S_3 for XML data in Figure 3.4. Note that the relaxed queries S_2 and S_3 have different set of free variables from those of Q and S_1. To ensure the "superset" property of query relaxation, thus, the notion of set containment needs to be extended properly to allow comparison between relations with different arity. We omit the details in the interest of the space and readers may refer to [LS02].

3.4.4 Order Relaxation

So far, we have ignored the effects of "order" in XML model. In the ordered XML model where the order among siblings is significant, an XML query carries another constraint on the order. Then, one can also *relax* the order among the siblings of the given query tree. For instance, in the ordered XML model, the query Q of Figure 3.5(c) is to find all occurrences of the tag `<a>` that has a descendant `<b>` *first* and a child `<c>` *second*. The added symbol "<" in the middle emphasizes the ordered semantics. Therefore, in Figure 3.5, the XML data D_1 can be an answer for Q, but not D_2 since the order is reversed. However, relaxing the order between tags `<b>` and `<c>` as in the query O of Figure 3.5(d), D_2 can be an approximate answer to Q. In $\mathcal{QAC}$ framework, using the function



```
<a>
    <d>
        <b/>
    </d>
    <c/>
</a>
```

(a) XML data D_1

```
<a>
    <c/>
    <d>
        <b/>
    </d>
</a>
```

(b) XML data D_2

(c) Original query Q

(d) Order-relaxed query O

Figure 3.5: Example of order relaxation.

$id(\$v)$ that returns the id of the variable $\$v$, the original and relaxed queries can be expressed as follows:

$$\begin{aligned}
\mathcal{Q}(\$1,\$2,\$3) &\equiv node(\$1,a) \wedge node(\$2,b) \wedge node(\$3,c) \wedge \\
&\quad ad\_edge(\$1,\$2) \wedge pc\_edge(\$1,\$3) \wedge id(\$2) < id(\$3) \\
O(\$1,\$2,\$3) &\equiv node(\$1,a) \wedge node(\$2,b) \wedge node(\$3,c) \wedge \\
&\quad ad\_edge(\$1,\$2) \wedge pc\_edge(\$1,\$3)
\end{aligned}$$

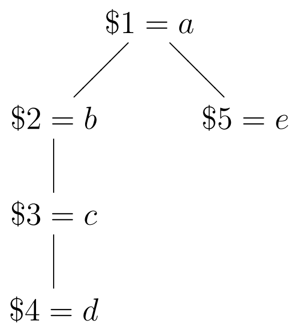
More extended notion of order relaxation, where the order of nodes in the hierarchy can be reversed in a vertical direction, appears in [KS01]. This too can be easily captured in $\mathcal{Q}AC$ framework and is omitted.

3.5 Relaxation Control

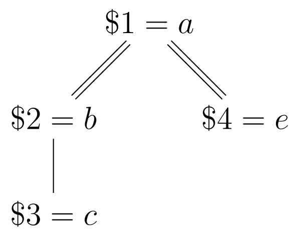
Given an original query Q, there are many ways to relax the Q. Previous works in [Cha90, KS01, ACS02] propose some evaluation algorithms, but does not provide a way to tune the relaxation process in a very detailed manner. Ideally, one wants to control the way/order of relaxation process. In $\mathcal{QAC}$ framework, the 3rd component, control statement, is provided to serve such purposes. For instance, consider an original query Q of Figure 3.6. Suppose one wants to control the relaxation process such that Q is relaxed in the order of R_1, R_2 and R_3. Furthermore, the user wants to make sure to have at least 1 answer tuple having "10" and "30" bound to variables \$1 and \$3, respectively. Then, the original query Q can be captured as follows in the $\mathcal{QAC}$ framework: $Q \equiv (\mathcal{Q}_1, \mathcal{A}_1, \mathcal{C}_1)$, where

$$\begin{aligned}
\mathcal{Q}_1(\$1, \$2, \$3, \$4, \$5) \quad &\equiv \quad N_1 : node(\$1, a) \wedge N_2 : node(\$2, b) \wedge N_3 : node(\$3, c) \wedge \\
&\quad\quad N_4 : node(\$4, d) \wedge N_5 : node(\$5, e) \wedge \\
&\quad\quad E_1 : pc\_edge(\$1, \$2) \wedge E_2 : pc\_edge(\$2, \$3) \wedge \\
&\quad\quad E_3 : pc\_edge(\$3, \$4) \wedge E_4 : pc\_edge(\$1, \$5) \\
\mathcal{A}_1(\langle \mathcal{Q}_1 \rangle) \quad &\equiv \quad \langle \mathcal{Q}_1 \rangle \ni (10, \_, 30, \_, \_) \\
\mathcal{C}_1 \quad &\equiv \quad \neg N_1 \wedge [(E_1 E_4 N_1 N_2 N_3 N_4)(E_3)(E_2)]
\end{aligned}$$

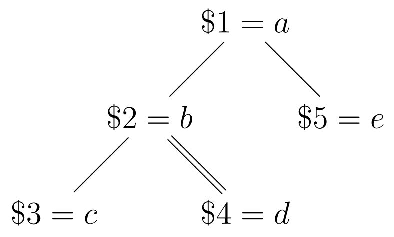
The control statement $\mathcal{C}_1$ states that the root node of the query should not be relaxed and when the relaxation occurs, the conjuncts labeled $E_1, E_4, N_1, N_2, N_3,$ and N_4 must be relaxed prior to the conjuncts labeled E_3, which in turn must be relaxed prior to the conjuncts labeled E_2 (assuming that the execution semantics inside of "()" construct is *concurrent* while that between "()" constructs is *sequential*). According to this rule, R_1 (i.e., edge relaxed query) of Figure 3.6 is the preferred relaxed query to users than R_2 or R_3 (i.e., subtree promotioned queries).



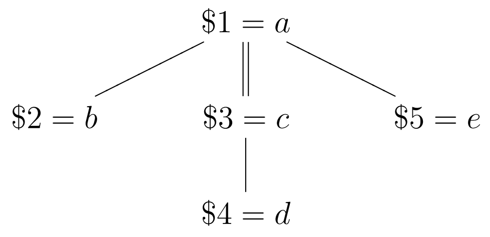
(a) Original query Q



(b) Relaxed query R_1



(c) Relaxed query R_2



(d) Relaxed query R_3

Figure 3.6: Example of three query relaxations.

3.6 Ranking

In returning a set of approximate answer set to users, it is important to *rank* answers based on their similarities to the original query so that more similar (thus better) answers are ranked higher than others. Therefore, given a query Q and its relaxed answer A, devising a useful ranking measure important.

In its abstract notion, consider a query $Q : /A/B$ that looks for a `B` that is a child of `A`. Suppose the following 3 answers are returned:

```
(1): <A><B/></A>
(2): <A>
(3): <A><C><B/></C></A>
```

Answer (1) is the exact match, while answers (2) and (3) are the relaxed matches.

It is clear that the match score of the answer (1) should be "1" due to its exact match with Q. However, exact score values for answers (2) and (3) are not clear and inherently arbitrary. If application semantics prefer answers with "optional" nodes to answers with "descendent" edges, for instance, then answers (2) will have a higher match score. Since devising a generic match score and a ranking scheme are closely related to the problem of distance measure between two XML data trees, this issue will be further covered in Chapter 4.

As one approach to ranking problem, one may assume that user's query provide tips regarding scores. For instance, imagine an XML query in terms of an abstract tree where (1) every node has XML tag name, and (2) every node and edge have a 3-tuple vector $v_i = (\text{max}_i, \text{min}_i, \text{method}_i)$, where

- max_i: The highest match score for an node/edge. This is the score for an exact match.
- min_i: The lowest match score for an node/edge. This is the lower bound for an relaxed match.
- method_i: User-defined method that decides how to assign a match score in the range of max_i and min_i: $m(v_i) = \text{method}_i(\text{max}_i, \text{min}_i)$.

and (3) a normalization factor Δ that enables the distance between 0 and 1 range. By default, Δ is the sum of all max_i values in Q. Therefore, for each node or edge, there is a match score calculated according to the method_i and denoted as $m(v_i)$. Overall match score M for Q, $M(Q)$, is then defined as follows:

$$M(Q) = \frac{1}{\Delta} \sum_{\forall v_i} m(v_i)$$

Match scores for the above three relaxation answers are thus calculated easily and illustrated in Figure 3.7. Assume that b refers to a "binary function"

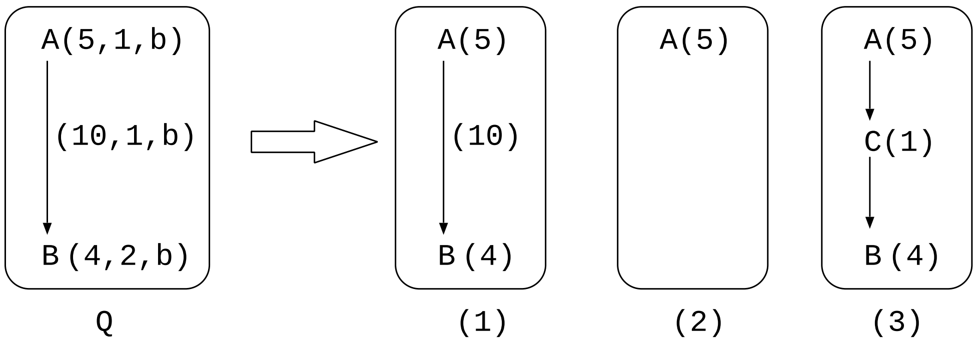


Figure 3.7: Example of three *path* query relaxations.

which chooses max or min value for an exact or relaxed match, respectively. Then, corresponding match scores for three cases in Figure 3.7 are following: (1) $M_1(Q) = \frac{1}{19}(5 + 10 + 4) = 1.0$, (2) $M_2(Q) = \frac{1}{19}(5) = 0.26$, and (3) $M_3(Q) = \frac{1}{19}(5 + 1 + 4) = 0.53$. As a conclusion, the final ranking among three answers would be as follows (similarity value is shown at first):

```
1.00: <A><B/></A>
0.53: <A><C><B/></C></A>
0.26: <A>
```

3.7 Summary

In this Chapter, we have proposed a framework, *QAC*, to describe the exact semantics and behavior of query relaxation for XML model. The benefits of such formalism are that it is both precise and concise. We also presented a variety of novel query relaxations of the XML model that have been proposed recently in [Cha90, CYC96, CG99, KS01, ACS02] and showed that *QAC* can not only capture their semantics precisely, but also describe their relaxation behavior.

Many interesting directions are ahead. First, designing and implementing an XML query language that is capable of expressing the aforementioned query relaxations is an important task in future. Second, $\mathcal{Q}AC$ framework currently lacks the fine control to dictate which relaxation is to occur in certain steps. Third, devising an efficient algorithm to evaluate a given set of relaxed queries in the context of $\mathcal{Q}AC$ framework is also an interesting problem.

CHAPTER 4

Distance Metric for XML Data Trees

To systematically enlarge query scope, a knowledge structure generated from the database can be used as a guidance. The benefits of such structure-based query relaxation, compared to a mechanical process of dropping certain sub-queries from the original query, is that it tends to be more intuitive for the given application domain. The fundamental issue to build such a knowledge structure is to devise a good *distance metric* (or *similarity metric*) that is used in the underlying data clustering process. Unlike in relational databases where atomic objects are tuples, in XML model, distance metric must be defined on the new type of atomic objects, tree-shaped XML data. Due to the inherent structural complexity and ill-defined "closeness" in XML model, however, devising an intuitive metric is not a trivial task, if at all possible. In this Chapter, we discuss some of the preliminary results on devising a good distance metric between tree-shaped XML data.

4.1 Background

Query relaxation is a process to enlarge the search scope for finding more *relevant* answers. Enlarging a query scope can be accomplished by viewing the queried object at different conceptual levels. Towards this problem, one technique, called TAH (Type Abstraction Hierarchy), has been developed in CoBase project [CYC96, CCH96] and successfully used in many applications [CYC94,

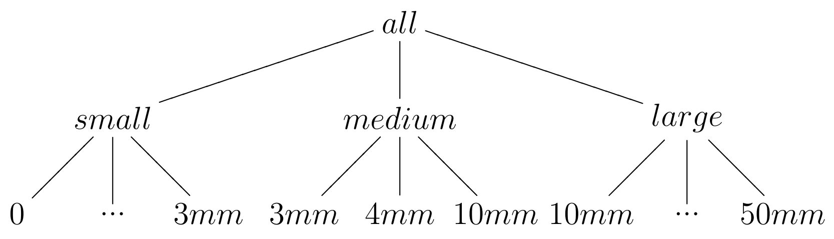


Figure 4.1: Example of TAH for brain tumor size.

CHC98].

TAH is a hierarchical (tree-like) data structure that can be automatically generated from database. Based on the values and frequencies information, data are clustered into a tree-like structure of TAH based on minimizing the average pairwise distance of instances among clusters. A problem of relaxing a query condition can then be reduced to that of finding the right range value that is represented by an appropriate cluster in the TAH. For instance, the query "Find medical image (e.g., MRI) with tumor size = 5 mm" can be relaxed to "Find medical image with tumor size between 3 mm and 10 mm", when the value range 3 mm to 10 mm is derived from the TAH of Figure 4.1. That is, a cluster called "medium" is the one appropriate for "4 mm" tumor size.

The fundamental building block needed to build such a TAH from databases is measuring the distance between two atomic objects and group the similar objects into the same cluster. Therefore, devising a good distance metric between objects plays a major role. In CoBase system, *Relaxation Error* is used to serve that purpose. Relaxation error is defined as the average difference between the requested values in the query and the returned values in the answer. Thus, from the query relaxation perspective, two objects are similar if relaxation error between them is small.

4.2 Related Work

General survey on data clustering and classification can be found in [AHD96]. Here, we focus on works related to distance metrics for non-traditional data types such as trees. The most well-known distance measure for trees is *Tree Edit Distance* and has been extensively studied (e.g., [Tai79, ZS89, CRG96, BCD95, WDC01, CAM02, GJK02, CTZ01]). In tree edit distance problem, a distance between two trees is defined as the summation of the costs needed to convert the source tree to target tree using the pre-defined set of edit operations such as insert or delete.

Since general tree edit distance problem (also known as *Tree-to-Tree editing problem* or *Tree-to-Tree Correction problem*) for unordered trees is known as NP-hard, works in [Tai79, ZS89, CRG96, WDC01] instead try to find an efficient algorithm for limited cases such as ordered/binary trees or trees with additional constraints. The best known algorithm for computing tree edit distance between two "ordered" trees is by Zhang and Shasha [ZS89] and has the time complexity of roughly $O(n^4)$, where n is the number of the nodes in a tree. Using the Zhang and Shasha's algorithm or other its variants (e.g., [Tai79, CRG96, BCD95, WDC01, CAM02, GJK02, CTZ01]), one can define the distance between two trees as its tree edit distance. Since XML model can be easily captured as trees, results developed for tree edit distance can be naturally adopted for distances for XML models. For instance, in a more recent work, [NJ02] presents a tree edit distance based measure with an application for clustering XML documents. Their focus is to group XML documents generated from different XML schemas by clustering.

As an another approach, one may transform non-traditional data types into traditional ones (e.g., set or bag model, vector-space model) and compute distances on the new types. For instance, in [GGW01], new measures that exploit a hierarchical domain structure in order to produce more intuitive similarity scores are presented. In the same spirit as [GGW01], we investigate several such measures for XML model in this Chapter.

4.3 Relaxation Index

To support query relaxation in XML model, we need a knowledge representation similar to TAH that guides the XML query relaxation process. Due to the hierarchical nature of XML data, it is more complex and difficult in automatically generating such structures for XML model.

We propose to develop a technique to generate relaxation index structures, XML Type Abstraction Hierarchy (X-TAH). X-TAH is a hierarchical tree-like knowledge structure that contains a summary of XML data trees. Intuitively, X-TAH has a hierarchical cluster where objects in the same cluster are *closer* to each other. Much like that objects are tuples in relational model, objects are trees in XML model. X-TAH has two node types – internal and leaf nodes – as shown in Figure 4.2.

- An internal node has a *representative tree* that summarizes the characteristics of the all data trees in that cluster. For instance, in Figure 4.2, the representative tree `R1` represents all the data trees of their descendants. In the query relaxation process, the system examines all the representative trees to find the one that contains the exact or the most similarly matching target.
- A leaf node has a representative tree as well as a set of *data trees* that are *similar* to each other with respect to their values and structures. For instance, in Figure 4.2, the three data trees with ids 1, 2, and 21 are closer

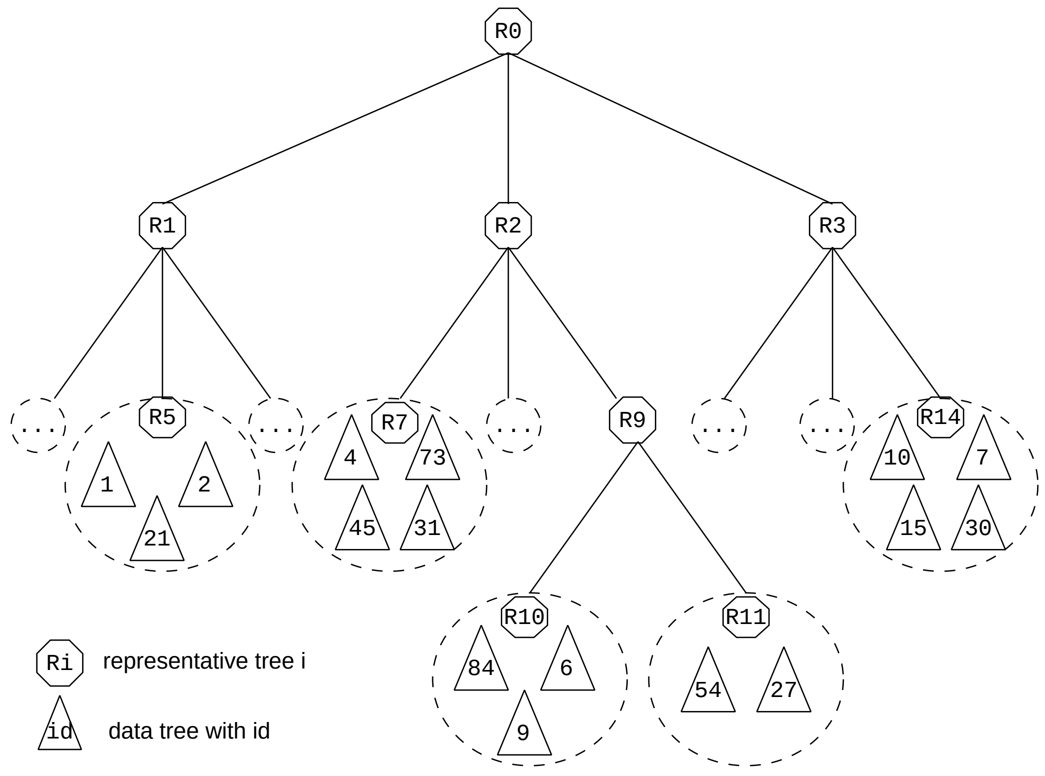


Figure 4.2: Example of X-TAH.

to each other than the trees with id 4 or 73. Hence, for a given query q, the system finds a data tree t_d in a representation tree cluster c that best matches q. The query q can then be relaxed to the neighbors of t_d which are candidates for the "next-best" answers.

X-TAH can be constructed using either agglomerative or divisive clustering algorithms. CoBase system has developed a family of efficient and effective agglomerative (e.g., ICE [ZC00]) as well as divisive (e.g., DISC [CCH96, CYC96]) clustering algorithms. For instance, for the DISC algorithm, initially, all n objects (i.e., a tuple in the relational model and a tree in XML model) belong to a single cluster. The number of clusters is increased at each step of the algorithm, by dividing one of the existing clusters into two offspring clusters. For the ICE algorithm, given n data trees, or n initial clusters which each containing one data tree, the algorithm merges the two closest clusters at each step, and a binary-tree can be constructed after the $(n-1)$-th iteration.

These algorithms perform clustering based on two distance metrics, *inter-object distance (d_{io})* and *inter-cluster distance (d_{ic})*. Since the inter-cluster distance is orthogonal to the characteristics of the underlying data model, the main difficulty lies in devising a plausible inter-object distance metrics between XML data trees. This is not a straightforward task since objects involved are not flat, but hierarchical. Such metrics need to be designed judiciously so that final X-TAH makes sense for the given XML data and application.

4.4 XML Inter-Cluster Distance (d_{ic})

Let us defer the discuss of inter-object distance to Section 4.5 and assume such a distance metric is given in this Section. Once an inter-object distance metric, d_{io}, for XML model is developed, inter-cluster distance metric can be used, for instance, to group the similar objects based on d_{io} and generate a hierarchical cluster. Since the data type can be a mix of numerical and categorical, the following inter-cluster error measure [ZC00] can be used as an inter-cluster distance to calculate the distance between two clusters C_1 and C_2

$$d_{ic}(C_1, C_2) \;=\; \sum_{(o_i)\in C_1} \sum_{(o_j)\in C_2} P(o_i|C_1)\; P(o_j|C_2)\; d_{io}(o_i, o_j) \tag{4.1}$$

where $P(o_i|C_k)$ is the conditional probability of the object o_i in C_k $(k = 1, 2)$, ie., $P(o_i|C_k) = \frac{o_i}{|C_k|}$, $|C_k| = \sum_{(o_i)\in C_k} o_i$ and $d_{io}(o_i, o_j)$ is an inter-object distance metric between two objects o_i and o_j.

The algorithm will generate a binary tree hierarchy, X-TAH, based on minimizing the inter-cluster distances between C_1 and C_2. Computing inter-cluster distance involves the vast number of initial clusters. For instance, the DISC algorithm has a time complexity of $O(n \log n)$, where n is the number of initial objects to cluster. Now, in next Section, we focus on the core of the problem of this Chapter – inter-object distance for XML model.

4.5 XML Inter-Object Distance (d_{io})

The distance metrics between two "tuples" in relation data are well defined whether or not they are numeric, categorical, or mixed types. However, because of the hierarchical structure of XML data, such distance metrics are rather complex and should have the following properties:

- The metric should capture distance among XML data trees with such aspects as values and/or structures of data trees, orders of siblings, depths of sub-elements, types of nodes (e.g., element, attribute, `CDATA`), etc.
- The metric should be adjustable by users to compute distances based on a selected set of aspects for a specific application domain.

Towards this goal, we consider two types of methods. First, in **Set Based Metrics**, each XML data tree is transformed into a set of objects that capture the main characteristics of the original XML data trees and two such sets are compared to compute distance afterwards. Second, in **Tree Edit Distance Based Metrics**, a distance is the cost of converting the source tree to the target tree. A traditional string edit distance is a metric between two strings, possible of unequal lengths, given by the minimum number of symbol insertions and deletions required to transform one string into the other. Tree edit distance is a generalization of the traditional string edit distance.

Let us consider a few proposals. For a simpler discussion, let us assume that there is a straightforward relationship between similarity s and distance d (e.g., $s + d =$ constant) [AHD96] so that once one value is computed, the other value can be also computed easily.

4.5.1 Set Resemblance Method

In this scheme, the similarity is defined in terms of the ratio of intersection over union of the sets. First, consider the following definition.

Definition 3 (Edge Constraint) *$u \xrightarrow{l} v$ is an* edge constraint *from node u to v with a label l. Two edge constraints $u \xrightarrow{l} v$ and $u' \xrightarrow{l'} v'$ are* matched *if $u = u'$, $l = l'$, and $v = v'$. Given two trees X and Y, e^X and e^Y are the set of edge constraints in X and Y, respectively.* □

Then, one may view the tree as a set of edge constraints and define the similarity between two trees with respect to their edge constraints. That is, two trees become more similar as they share more common edge constraints.

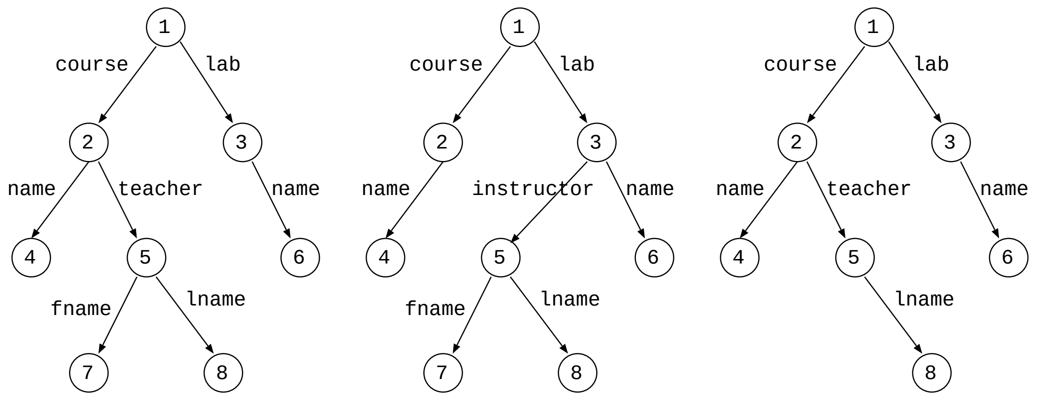
$$sim(X, Y) = \frac{|e^X \cap e^Y|}{|e^X \cup e^Y|} \tag{4.2}$$

Example 1. Consider three trees A, B, and C in Figure 4.3. Edge constraint set for them are the following:

$$\begin{aligned}
e^A &= \{1 \xrightarrow{course} 2, 1 \xrightarrow{lab} 3, 2 \xrightarrow{name} 4, 2 \xrightarrow{teacher} 5, 3 \xrightarrow{name} 6, 5 \xrightarrow{fname} 7, 5 \xrightarrow{lname} 8\} \\
e^B &= \{1 \xrightarrow{course} 2, 1 \xrightarrow{lab} 3, 2 \xrightarrow{name} 4, 3 \xrightarrow{instructor} 5, 3 \xrightarrow{name} 6, 5 \xrightarrow{fname} 7, 5 \xrightarrow{lname} 8\} \\
e^C &= \{1 \xrightarrow{course} 2, 1 \xrightarrow{lab} 3, 2 \xrightarrow{name} 4, 2 \xrightarrow{teacher} 5, 3 \xrightarrow{name} 6, 5 \xrightarrow{lname} 8\}
\end{aligned}$$

Using Equation 4.2, similarities among them can be defined as follows:

$$sim(A, B) = \frac{|e^A \cap e^B|}{|e^A \cup e^B|} = \frac{6}{7}$$



(a) Tree A (b) Tree B (c) Tree C

Figure 4.3: Example of similar XML data trees.

$$sim(A, C) = \frac{|e^A \cap e^C|}{|e^A \cup e^C|} = \frac{6}{7} \qquad \square$$

Clearly, Equation 4.2 exhibits some problems; it cannot differentiate $sim(A, B)$ and $sim(A, C)$. This is because Equation 4.2 does not consider the structural characteristics of the trees. To remedy this problem, one needs to take a notion of path (i.e., root to leaf path) into consideration as follows.

Definition 4 (Path Constraint) *A* path constraint *is a unique concatenation of edge constraints $x_1 \xrightarrow{l_1} x_2$, $x_2 \xrightarrow{l_2} x_3$, ... $x_{n-1} \xrightarrow{l_{n-1}} x_n$, where x_1 is a root node and x_n is a leaf node, and denoted as $x_1 \xrightarrow{l_1} x_2 \xrightarrow{l_2} ... \xrightarrow{l_{n-1}} x_n$. Given two trees X and Y, p^X and p^Y are the set of path constraints in X and Y, respectively.* $\square$

Based on the definition of path constraint, instead of edge constraint, one can define the similarity metric as follows:

$$sim(X, Y) \;=\; \frac{|p^X \cap p^Y|}{|p^X \cup p^Y|} \qquad (4.3)$$

Example 2. Consider three trees A, B, and C in Figure 4.3 again. Path constraint sets for them are the following:

$$\begin{aligned}
p^A &= \{1 \xrightarrow{course} 2 \xrightarrow{name} 4, 1 \xrightarrow{course} 2 \xrightarrow{teacher} 5 \xrightarrow{fname} 7, 1 \xrightarrow{course} 2 \xrightarrow{teacher} 5 \xrightarrow{lname} 8, \\
&\quad 1 \xrightarrow{lab} 3 \xrightarrow{name} 6\} \\
p^B &= \{1 \xrightarrow{course} 2 \xrightarrow{name} 4, 1 \xrightarrow{lab} 3 \xrightarrow{instructor} 5 \xrightarrow{fname} 7, 1 \xrightarrow{lab} 3 \xrightarrow{instructor} 5 \xrightarrow{lname} 8, \\
&\quad 1 \xrightarrow{lab} 3 \xrightarrow{name} 6\} \\
p^C &= \{1 \xrightarrow{course} 2 \xrightarrow{name} 4, 1 \xrightarrow{course} 2 \xrightarrow{teacher} 5 \xrightarrow{lname} 8, 1 \xrightarrow{lab} 3 \xrightarrow{name} 6\}
\end{aligned}$$

Using Equation 4.3, similarities among them can be defined as follows:

$$sim(A,B) = \frac{|p^A \cap p^B|}{|p^A \cup p^B|} = \frac{2}{6}$$

$$sim(A,C) = \frac{|p^A \cap p^C|}{|p^A \cup p^C|} = \frac{3}{5} \qquad \square$$

Path-constraint based set scheme in Equation 4.2 is superior to the edge-constraint based set scheme in Equation 4.3 in that it retains one kind of structural constraint (i.e., parent-child relationship) of the tree through path constraints. However, since path-constraint based set scheme treats only the whole root-to-leaf path as an atomic unit, it cannot handle subtle difference between two similar paths when one is a prefix of the other. That is, the scheme will find two paths *a.b.c.d* and *a.b* as simply “different” despite their similarities from the root *a* upto the second node *b*. Next scheme aims at overcoming this shortcoming.

4.5.2 Prefix-Clustered Similarity Method

Let us consider other metric that takes the “depth” of tree structure into consideration. People have developed a technique that maps strings to a linear space in a manner that ensures strings that are extensions of a particular prefix are clustered, while preserving lexicographic ordering [JKS00]. One can use such a technique to convert string values (i.e., node label) of tree into some numeric values and use conventional similarity metric comparing the values.

Let us briefly describe the technique in [JKS00].

> *"... Let the size of the alphabet be α, with an established lexicographic order on the symbols in the alphabet. Choose an integer $\beta > 2\alpha$. Let a string of length n be $s_1 s_2 ... s_n$, with each symbol s_i mapped to an integer t_i between 1 and α. The string as a whole is then mapped to $t_1/\beta + t_2/\beta^2 + ... + t_n/\beta^n$. Let us look at what is going on with an example, choosing β to be $2\alpha + 1$. In this case, let $[i; j]$ be the interval between two strings of some (equal) length that are adjacent to each other in the lexicographic ordering. Then, extensions of the first string by one character are equally placed in the open interval $[i; (i + j)/2]$. Thus, all extensions of the first string are still closer to i than to j in value. The role of β is to determine the "margin of victory". While it is technically sufficient to set β to be marginally greater than 2α, [JKS00] suggests that a value of $\beta = 2\alpha + 1$ be sufficient..."*

By viewing a *path* that consists of an array of concatenated node labels as a *string* that consists of alphabets, one can utilize the above technique to map the path string into numeric space with little modification as follows:

1. Given a tree X, gather all unique node labels into sets T^X. Assume that there is function $t_X(l)$ that returns an established order of node label l.

2. Map a path constraint $p_i = a_1 a_2 ... a_n$ into some rationale number

$$M(p_i) = t_X(a_1)/\beta + t_X(a_2)/\beta^2 + ... + t_X(a_n)/\beta^n$$

After each path of trees is mapped to numeric space, one can define the similarity of two trees X and Y in terms of average pair-wise Euclidean distance as follows:

$$sim(X,Y) \quad = \quad 1 - (\sum_{k,l=1}^{m} (|M_A(p_k) - M_B(p_l)|)^2)^{\frac{1}{2}} \tag{4.4}$$

Example 3. Consider trees A and B in Figure 4.3 again. Then, we have the alphabet space of $T = \{course, fname, instructor, lab, lname, name, r, teacher\}$ with r being the root node label and with a lexicographic order among them (e.g, $t(course) = 1$, $t(fname) = 2$, etc). $\alpha = 8$ and $\beta = 2\alpha + 1 = 17$. Each tree has the following paths, respectively.

$$\begin{aligned} p^A &= \{r.course.name, r.course.teacher.fname, r.course.teacher.lname, \\ &\quad r.lab.name\} \\ p^B &= \{r.course.name, r.lab.instructor.fname, r.lab.instructor.lname, \\ &\quad r.lab.name\} \end{aligned}$$

Based on these, the path can be mapped to numeric values as follows:

$$\begin{aligned} M_A(r.course.name) &= t(r)/\beta + t(course)/\beta^2 + t(name)/\beta^3 \\ &= 7/17 + 1/17^2 + 6/17^3 \\ M_A(r.course.teacher.fname) &= t(r)/\beta + t(course)/\beta^2 + t(teacher)/\beta^3 \\ &\quad + t(fname)/\beta^4 \\ &= 7/17 + 1/17^2 + 8/17^3 + 2/17^4 \\ M_A(r.course.teacher.lname) &= t(r)/\beta + t(course)/\beta^2 + t(teacher)/\beta^3 \\ &\quad + t(lname)/\beta^4 \\ &= 7/17 + 1/17^2 + 8/17^3 + 5/17^4 \\ M_A(r.lab.name) &= t(r)/\beta + t(lab)/\beta^2 + t(name)/\beta^3 \end{aligned}$$

$$= 7/17 + 4/17^2 + 6/17^3$$

Similarly,

$$\begin{aligned}
M_B(r.course.name) &= t(r)/\beta + t(course)/\beta^2 + t(name)/\beta^3 \\
&= 7/17 + 1/17^2 + 6/17^3 \\
M_B(r.lab.instructor.fname) &= t(r)/\beta + t(lab)/\beta^2 + t(instructor)/\beta^3 \\
&\quad + t(fname)/\beta^4 \\
&= 7/17 + 4/17^2 + 3/17^3 + 2/17^4 \\
M_B(r.lab.instructor.lname) &= t(r)/\beta + t(lab)/\beta^2 + t(instructor)/\beta^3 \\
&\quad + t(lname)/\beta^4 \\
&= 7/17 + 4/17^2 + 3/17^3 + 5/17^4 \\
M_B(r.lab.name) &= t(r)/\beta + t(lab)/\beta^2 + t(name)/\beta^3 \\
&= 7/17 + 4/17^2 + 6/17^3
\end{aligned}$$

Now, using Equation 4.4, similarities between trees A and B can be defined accordingly. □

So far, we have considered three set-based metrics for XML model – edge constraint, path constraint, and prefix clustered similarity metrics. In the subsequent Section, we consider a variation based on tree edit distance.

4.5.3 Tree Edit Distance Method

The problem of computing the distance between two trees is a generalization of the problem of computing the distance between two strings to labeled trees. When dealing with *unordered* trees, this problem is known to be NP-complete [ZS89]. Thus, people have focused on finding restricted cases such as *ordered* or *degree-*

2 trees with which a notion of distance is meaningful for an application under consideration.

The traditional edit distance between two strings, of unequal lengths, is the minimum number of symbol edit operations (e.g., insert, delete, or update) required to transform from one string to the other. One can view the distance between two trees as a generalization of computing the distance between the two strings (e.g., [Tai79, ZS89]). By applying such a tree distance metric to XML model, the *similarity* between the specified query and approximate answer can be computed as the *distance* between the query tree and data tree. Similarly, relaxing a query q to the best approximate query q' is equivalent to mapping a tree t_q (a tree representation of the query q) to another tree $t_{q'}$ with minimum edit operation cost.

To use the edit distance based metric for query relaxation, one needs to incorporate domain specific *semantics* into the distance metric. Edit distance based algorithms such as Z&S algorithm [ZS89] does not consider the cost assigned to each operator. Whether assigning equal costs to all the operators (i.e., $cost(insert) = cost(delete) = cost(update) = \sigma$) or variable costs (i.e., $cost(insert) = \alpha$, $cost(delete) = \beta$, and $cost(update) = \gamma$) to the operators does not affect the correctness of the algorithms. Equal cost method, however, does not work well because of the semantics that may be present in the tree structure. For instance, in XML model, “inserting” a node near the root node may be more significant than “deleting” a node near the leaf nodes. Therefore, we develop a methodology to assign variable cost to the operator based on the context of a given application domain. Using a variable cost edit distance metrics is more intuitive, but greatly increases the complexity. There has been very little work, if any, done in this area.

In short, inter-object distance, d_{io}, between two tree objects A and B is defined as the summation of the minimal operation costs that are needed in order to convert the source tree A to the target tree B:

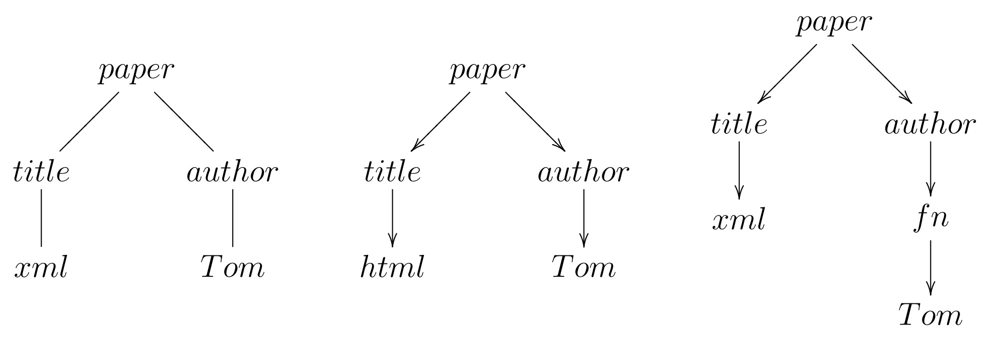
$$d_{io}(A,B) = \sum_{i,j} cost(op_i(v_j)) \tag{4.5}$$

where the operator op_i is applied to the node v_j of the source tree A. Further, let $\Lambda(op_i)$ and $\Lambda(v_j)$ be the characteristics of the operator op_i and node v_j, respectively. Then, $cost()$ in Equation 4.5 can be rephrased to:

$$cost(op_i(v_j)) = \frac{1}{\Delta}(\sum_{\forall i} W_i \Lambda_i(op_i) + \sum_{\forall j} W_j \Lambda_j(v_j)) \tag{4.6}$$

where W_i and W_j are weight factors to reflect the relative importance of the characteristics of operators and nodes in the formula, respectively and Δ is a normalization factor. Most of the existing works using tree edit distance assume the cost of operations to be a constant α. On the contrary, The formula for $cost(op_i(v_j))$ provides a generic method to assign arbitrary costs to operators differently using their characteristics. There are various types of operators and node characteristics that may affect the cost assignment. We review a few examples below:

1. **Operator Type**: The basic Z&S algorithm [ZS89] assigns equal cost to all the operators. We shall assign costs based on the operator. For example, using equal cost scheme in Figure 4.4, both data trees d_1 and d_2 will return approximate answers with equal distances σ, even for the two different types of operations. In this example, the distances between q and d_1, $dist(q, d_1)$, and the distance between q and d_2, $dist(q, d_2)$, are identical; that is, $dist(q, d_1) = cost(update(xml \rightarrow html)) = \sigma$ and $dist(q, d_2) = cost(insert(fn)) = \sigma$. Variable cost assignment allows us to differenti-



(a) Query q (b) Data tree d_1 (c) Data tree d_2

Figure 4.4: Example of query answers using equal operator cost scheme.

ate one data tree from the other and mark one as a "better" approximate answer.

2. **Node Level**: The costs for an operator can be dependent on the level of the node that the operator is applied. The intuitive idea of this variation is that editing the root node and editing leaf node may yield significantly different impact. Assuming the operation near the root node is more important (thus more expensive to modify) than the ones further down from the root node, the following formula, for instance, can be used to assign a variable cost:

$$cost(v) = \frac{1}{level(v)^k}$$

where $k(\geq 1)$ is a relevant factor. This formula assigns cost=1 when the root node is modified and 0 when the leaf node with infinite depth is modified. The example in Figure 4.5 illustrates how variable costs are assigned to nodes at the different level. In this example, assuming the relevant factor $k = 2$, then $cost(cite) = \frac{1}{2^2} = 0.25$ in (b). Similarly, $cost(cite) = \frac{1}{3^2} = 0.11$ in (c). Therefore, the data tree d_2 is a "better" answer than the data tree d_1. Data mining techniques may be adopted using a training set to determine the parameter k for a given application domain.

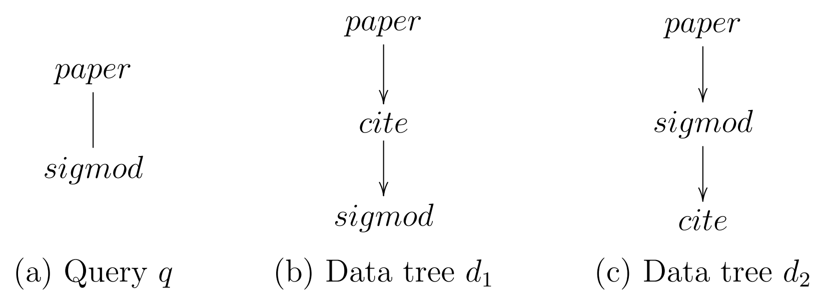


Figure 4.5: Example of node level cost adjustment.

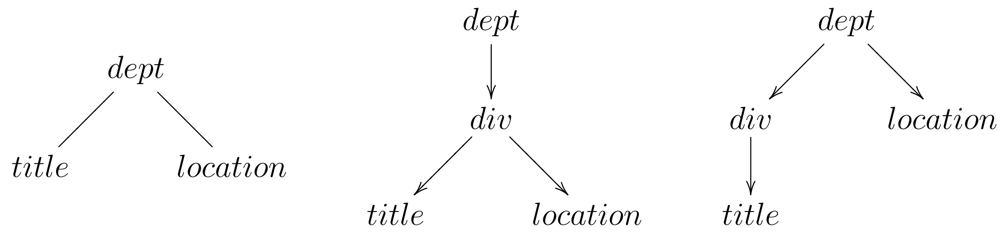
3. **Node Branching**: We shall use the branching factor b (i.e., number of children) of node v to differentiate their operation costs as follows:

$$cost(v) = b(v) = \frac{\text{number of children of } v}{\text{base of } v}$$

where *base* of v is either the number of maximum possible children of v that can be obtained from the XML schema [TBM01, BM01], or the average number of children of v obtained by mining the XML documents a priori. For instance, consider the following XML DTD describing a *dept* element:

```
<!ELEMENT dept    (code?,(div|(title,manager?))>
<!ELEMENT div     (title,location?)>
```

Note that the *dept* element has either *div* sub-element or a pair of *title* and *manager* sub-elements. Further, the *div* element has a mandatory *title* sub-element and an optional *location* sub-element. Thus the number of maximum possible children of *dept* and *div* are 3 and 2, respectively. In Figure 4.6, if the base case = 2, then $b(div) = \frac{2}{2} = 1$ for (b) (i.e., *div* in d_1 has 2 children) and $b(div) = \frac{1}{2} = 0.5$ for (c) (i.e., *div* in d_2 has 1 child). Thus modifying the *div* node in d_1 may be considered as more expensive than one in d_2.

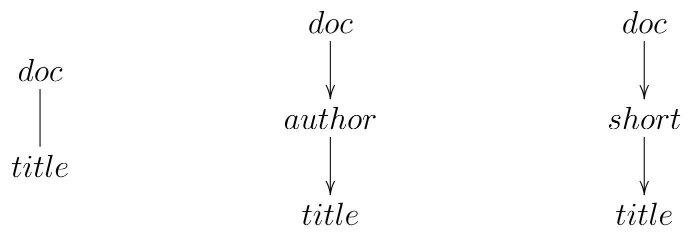


(a) Query q (b) Data tree d_1 (c) Data tree d_2

Figure 4.6: Example of node branching with different branching factors.

4. **Semantic Interpretation**: The semantics of the node in the context may affect the cost of the operation. For instance, consider the example in Figure 4.7, where two data trees yield the same cost based on node type, node level, and node branching. In this example, the *title* in d_1 may have a value (i.e., child node) "professor" while the *title* in d_2 may have a value "XML Tutorial". The problem stems from the ambiguous usage of the same term *title* under different semantics.

   However, based on the context, one may know that the node "title" in tree d_1 describes the "job" title of the author such as professor or research scientist, while the node "title" in tree d_2 is similar to the node "title" in the query q that describes the "paper" title of the document, whether it is a short or long title of the document. To identify such semantic subtlety, one may use a training set to derive semantic sensitive operation rules via data mining. A list of rules such as "when *title* is used with *author*, its semantics are different from the case when it is used with *doc* (confidence=90%)" may be learned from the training set to adjust the cost assignments. Then, based on such rules, the node *author* will be assigned a higher cost than the node *short*. As a result, d_2 will be a better answer than d_1 against the query q.



(a) Query q (b) Data tree d_1 (c) Data tree d_2

Figure 4.7: Example of different semantic interpretation.

4.5.4 Learning Costs via Machine Learning

Suppose a tree edit distance algorithm such as Z&S algorithm found the distance of two XML trees, A and B, as the cost for a single operation of $insert(x)$. Furthermore, suppose the following four types of characteristics are considered in Equation 4.6.

1. Λ_1: the type of operator
2. Λ_2: the level of the node
3. Λ_3: the number of the children
4. Λ_4: the semantics of node

Assuming $\Lambda_1(insert) = 0.8$, $\Lambda_2(x) = 0.5$, $\Lambda_3(x) = 0.3$, and $\Lambda_4(x) = 0$, $W_1 = W_2 = W_3 = W_4 = 1$ (i.e., equal weighting) and $\Delta = 4$, the distance between A and B can be computed as follows:

$$\begin{aligned} d_{io}(A,B) &= cost(insert(x)) \\ &= \frac{1}{\Delta}(\sum_{i=1} W_i\Lambda_i(insert) + \sum_{j=2,3,4} W_j\Lambda_j(x)) \qquad (4.7) \\ &= \frac{1}{4}(1 \times 0.8) + \frac{1}{4}(1 \times 0.5 + 1 \times 0.3 + 1 \times 0) \\ &= 0.4 \end{aligned}$$

That is, the distance between two trees is 0.4. Note that if one used uniform costs assigned to each operations in Equation 4.5 (i.e., $cost(insert) = cost(delete) = cost(update) = 1$), then $d_{io}(A, B) = cost(insert(x)) = 1$, which is quite far from 0.4. This, this example demonstrates the importance of non-uniform cost assignment in the tree edit distance based metrics.

Thus, to be able to learn the most appropriate costs for the given application domain, we propose to use *Machine Learning* techniques [Mit97]. Specifically, we re-cast our cost assignment problem in Equation 4.6 to a machine learning program as follows:

1. First, assume that there be n training set instances I_1, ... I_n. Each instance I_i $(1 \leq i \leq n)$ contains 3-tuple:
   - Q_i: the query asked.
   - A_i: the approximate answer found.
   - L_i: a label {Yes, No}, which is "Yes" if A_i is "relevant" to Q_i, and "No" otherwise.
2. There is a some ideal target function V and its approximation function $\hat{V}$, which will be calculated as a linear combination of the features extracted from XML data and application domains. For instance,
   - Λ_1: the difference of the number of branches in Q_i and A_i.
   - Λ_2: the difference of the height in Q_i and A_i.
   - Λ_3: the difference of the number of nodes in Q_i and A_i.
   - ...
   - Λ_k

Then, the approximation function $\hat{V}$ for each training set instance can be calculated as follows:

$$\hat{V}(I_i) = W_1\Lambda_1 + W_2\Lambda_2 + ... + W_k\Lambda_k$$

Since Λ_i $(1 \leq i \leq k)$ can be obtained from the Q_i and A_i in I_i, this essentially amounts to the problem of learning values for the coefficients W_1 through W_k in the target function representation.

3. Assign an arbitrary start value as the ideal target value of $V(I_i)$. For instance,

$$V(I_i) = \begin{cases} +1 & \text{if } I_i \text{ has a label “Yes”} \\ -1 & \text{if } I_i \text{ has a label “No”} \end{cases}$$

4. For all training set instances I_1, ... I_n, update weights W_1, ... W_k iteratively using LMS (Least Mean Square) algorithm [WH60, Lin95] as follows:

   (a) Initialize each W_i to some small random value like 0.05.

   (b) For each I_i,

      i. Use the current W_i to calculate $\hat{V}(I_i)$ as

$$\hat{V}(I_i) = W_1\Lambda_1 + W_2\Lambda_2 + ... + W_k\Lambda_k$$

      ii. For each weight W_i, update it as

$$W_i \leftarrow W_i + \eta \times (V(I_i) - \hat{V}(I_i)) \times \Lambda_i$$

      where η is a small constant (e.g., 0.1) that moderates the size of the weight update.

Note that when the error $V(I_i) - \hat{V}(I_i)$ is zero, no weights are changed. When $V(I_i) - \hat{V}(I_i)$ is positive (i.e., when $\hat{V}(I_i)$ is too low), then each weight is increased

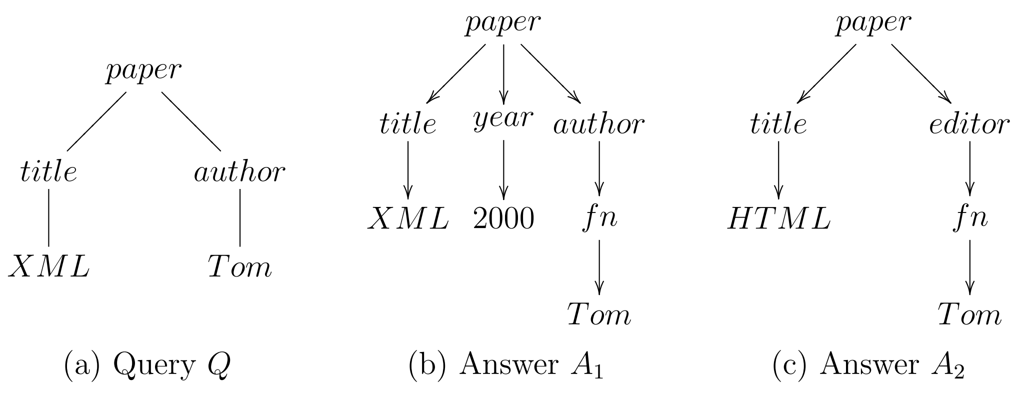


Figure 4.8: Example of a query and two approximate answers.

in proportion to the value of its corresponding feature. This will raise the value of $\hat{V}(I_i)$, reducing the error.

Example 4. Consider Figure 4.8. Suppose one obtained the following training set examples from Figure 4.4.

- $I_1 = (Q, A_1, Yes)$
- $I_2 = (Q, A_2, No)$

That is, the data A_1 in Figure 4.4 is found "relevant" to the query Q, while the data A_2 in Figure 4.4 is found "non-relevant" by some human experts. Further, let us assume that we consider the following three features extracted from the training set examples:

- Λ_1: the difference of the number of branches.
- Λ_2: the difference of the height.
- Λ_3: the difference of the number of nodes.

Also, assign either "+10" or "-10" as the starting value of $V(I_i)$. Then, the following initial training set instances can be obtained.

- $I_1 = (\langle \Lambda_1 = 1, \Lambda_2 = 1, \Lambda_3 = 3\rangle, +10)$
- $I_2 = (\langle \Lambda_1 = 0, \Lambda_2 = 1, \Lambda_3 = 1\rangle, -10)$

Now, initially we set all $W_i = 1$ and $\eta = 0.1$. Then,

- Iteration 1 for I_1:

$$\begin{aligned}
\hat{V}(I_1) &= W_1\Lambda_1 + W_2\Lambda_2 + W_3\Lambda_3 = 1 \times 1 + 1 \times 1 + 1 \times 3 = 5 \\
W_1 &= W_1 + \eta \times (V(I_1) - \hat{V}(I_1)) \times \Lambda_1 = 1 + 0.1 \times (10 - 5) \times 1 = 1.5 \\
W_2 &= W_2 + \eta \times (V(I_2) - \hat{V}(I_2)) \times \Lambda_2 = 1 + 0.1 \times (10 - 5) \times 1 = 1.5 \\
W_3 &= W_3 + \eta \times (V(I_3) - \hat{V}(I_3)) \times \Lambda_3 = 1 + 0.1 \times (10 - 5) \times 3 = 2.5
\end{aligned}$$

- Iteration 2 for I_2:

$$\begin{aligned}
\hat{V}(I_1) &= W_1\Lambda_1 + W_2\Lambda_2 + W_3\Lambda_3 = 1.5 \times 0 + 1.5 \times 1 + 2.5 \times 1 = 4 \\
W_1 &= W_1 + \eta \times (V(I_1) - \hat{V}(I_1)) \times \Lambda_1 = 1.5 + 0.1 \times (-10 - 4) \times 0 = 1.5 \\
W_2 &= W_2 + \eta \times (V(I_2) - \hat{V}(I_2)) \times \Lambda_2 = 1.5 + 0.1 \times (-10 - 4) \times 1 = 1.1 \\
W_3 &= W_3 + \eta \times (V(I_3) - \hat{V}(I_3)) \times \Lambda_3 = 2.5 + 0.1 \times (-10 - 4) \times 1 = 2.1
\end{aligned}$$

When enough training set instances are provided, weights W_i will continue to be changed and eventually be converged. Once the weights W_i are converged, finally, they may be fed into Equation 4.7 to provide non-uniform cost effect. □

4.6 Summary

Fundamental issue in supporting query relaxation for XML model is how to assess the *distance* (or equivalently *similarity*) between the original user query and answer candidates. It is essential to filter out all those answers that are approximate answers quickly. Furthermore, it is also important to identify the most closest answers to the query as the best answer, and next one as the second best one, and so on. In this Chapter, towards these issues, we discussed how to measure the similarity between two trees, where one is the user query and the other is the answer candidate. We especially studied two notions of distance (or similarity) metrics: set-based and tree edit distance-based ones. Our study is only preliminary and requires more extensive investigation on the subject.

CHAPTER 5

Selectivity Estimation of Relaxed XML Queries

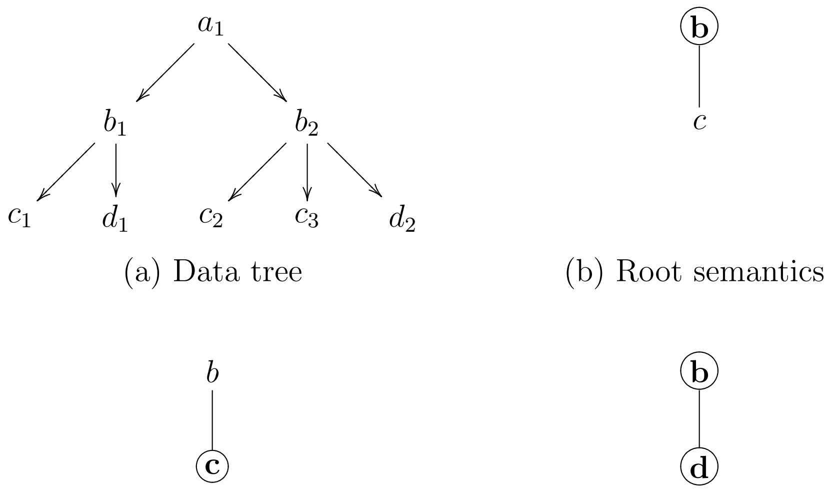
Due to the heterogeneity of XML data, it is often more useful to permit approximate matching of XML queries, in the spirit of Information Retrieval, than to return only exact answers. Query relaxation provides a natural basis for approximate matching of XML queries, where relaxed queries are obtained by structural transformations and type generalizations on the original query, such that all matches to the original query are included in the matches to any relaxed query. Estimating the number of relaxed twig matches is of use in providing user feedback, and in query optimization.

In this Chapter, we consider the problem of accurately estimating the number of answer matches, based on a given set of relaxed queries. Towards this problem, we propose an efficient method that (1) accurately computes selectivity estimates for each relaxed XML query, using a natural generalization of the correlated subpath tree (CST) summary structure, and (2) carefully combines these estimates by analyzing the nature of overlap between the different relaxed XML queries.

5.1 Background

We view the hierarchically organized data (e.g., XML documents) as node-labeled trees following [CJK01]. Then, a natural way to query such data is by using small node-labeled trees, referred to as **twigs**, that match portions of the hierarchical data. To see the motivation of the application that we deal with, suppose users are interested in finding: (1) all answers that approximately match the given twig beyond some threshold, or (2) top k answers ranked according to the similarity to the twig. A fundamental problem in this context is to accurately and quickly estimate the number of *approximate* matches of a twig query against the node-labeled data tree. This problem is relevant for providing users with quick feedback about their query, either before or along with returning query answers. Another use is in the cost-based optimization for efficiently evaluating such queries. In this Chapter, we do not consider issues of semantics of a relaxed twig or of efficiently finding approximate matches of a twig and refer interested readers to the papers [KS01, ACS02] for details. Our technical contributions in this Chapter are as follows:

- We extend a twig selectivity problem [CJK01] into a notion of *relaxed twig selectivity problem*. This is achieved by (1) determining the selectivity estimates of the involved relaxed twigs using the auxiliary data structure, and (2) carefully combining all those estimates such that matches overlapped among the relaxed twigs are not counted more than once to avoid over-estimation.
- We also propose generic projection-based query semantics that encompass several different semantics supported by different query languages (e.g., XPath [CD99], XQuery [CFR01]). Although we focus on the total semantics in this Chapter, since our technique is based on this generic notion, it can be readily adapted to the leaf or root semantics (see Figure 5.1).
- We show extensive experimental results that show the accuracy and robustness of our proposal for real XML data sets.



(a) Data tree (b) Root semantics

(c) Leaf semantics (d) Total semantics

Figure 5.1: Example of different query matching semantics.

Definition 5 (Twig Match) *The* twig match *of a twig query $Q = (V_Q, E_Q)$ in a node-labeled data tree $T = (V_T, E_T)$, denoted as $\langle Q(T) \rangle$ or simply $\langle Q \rangle$ when context is clear, is defined by a 1-1 mapping:*

$$f : V_Q \rightarrow V_T$$

such that if $f(u) = v$ for $u \in V_Q$ and $v \in V_T$, then (1) Label(u) = Label(v), and (2) if $(u, u') \in E_Q$, then $(f(u), f(u')) \in E_T$, where (u, u') is either a parent-child or ancestor-descendent edge and $(f(u), f(u'))$ preserves the same relationship. □

Note that the above twig match definition is essentially of a *total match*, where all the nodes of a twig query must be matched to non-null values. We observe that by relaxing this constraint, one can flexibly express a variety of match semantics that different XML query languages support. That is, by specifying only a subset of nodes in T, termed as the *projected nodes*, as significant, *partial match* can be supported. To illustrate this point, let us consider three most common semantics using an XML data tree in Figure 5.1, where projected nodes are circled.

- *Root Semantics*: Only root node is a projected node. For instance, an XPath [CD99] query "`//b[c]`" depicted in Figure 5.1.(b) returns $\{(b_1), (b_2)\}$ as a twig match. That is, users are only interested in the matching values of the node b, not c, so long as b has a child node c.

- *Leaf Semantics*: Only leaf node is a projected node. Similarly, an XPath query "`//b/c`" depicted in Figure 5.1.(c) returns $\{(c_1), (c_2), (c_3)\}$ as a twig match. For instance, a twig match in [AAN01] has the leaf semantics.

- *Total Semantics*: All nodes in a twig are projected nodes. An XQuery [CFR01] query "`FOR $b IN //b, $c IN $b/c RETURN $b,$c`" depicted in Figure 5.1.(d) returns $\{(b_1, c_1), (b_2, c_2), (b_2, c_3)\}$ as a twig match. As an another example, a twig match proposed in [CJK01] has a total semantics as well.

Throughout the rest of the Chapter, projected nodes in a twig are underlined for distinction (e.g., $/\underline{a}/b/\underline{c}$). A twig query P with projected nodes P_Q is denoted by a triple $P = (V_Q, E_Q, P_Q)$, where $P_Q \subseteq V_Q$.

Definition 6 (Projected Twig Match) *The* projected twig match *of a twig query $Q = (V_Q, E_Q, P_Q)$ in a node-labeled data tree $T = (V_T, E_T)$ is a twig match, where only mapping involving the projected nodes $f : P_Q \rightarrow V_T$ is significant.* □

Note that in the projected twig matches, all the mappings, whether projected or not, are relevant, but only the projected matches remain in the final result. In other words, we do not ignore the mapping for non-projected nodes during processing.

In order to explore the set of approximate matches of a query, one must be able to *relax* the query that gurantees that the set of answers to be returned is a superset of the set of exact query matches (i.e., no false dismissal). One may think of many ways to relax the given twig query: weakening condition values, swaping order of nodes, or deleting some nodes in a twig, etc. In this Chapter, we focus on the following two primitive relaxations, proposed in [ACS02][1]:

- *Node deletion*: A non-root node v in a twig can be deleted as follows: (1) if v is a leaf, simply v is deleted, and (2) if v is an internal node, then v is deleted and children of the node v become grand-children of v's parent node.

- *Edge relaxation*: A child edge (denoted by single edge "/") in a twig can be relaxed to a descendant edge (denoted by double edge "//"). Descendant edges cannot be relaxed further.

Definition 7 (Relaxed Twig) *A* relaxed twig *query $R = (V_R, E_R)$ is a twig obtained by applying one or many of the above relaxations to a twig $Q = (V_Q, E_Q)$.* $\Box$

If the function $sel(Q)$ returns a selectivity estimate of a twig Q, then, by definition, $\langle Q \rangle \subseteq \langle R \rangle$[2] and $sel(Q) \leq sel(R)$.

5.2 Related Work

Two areas in literature – query relaxation and selectivity estimation – are inevidently relevant to our work in this Chapter. Since works related to query relaxation are already surveyed in Chapter 3.2, here we only review works related to selectivity estimation in XML context.

[1]In fact, [ACS02] achieves deletion of non-leaf nodes using a combination of *subtree promotion* and *leaf node deletion*.

[2]The precise semantics of $\subseteq$ operator under the "projected" model will be precisely defined in Section 5.4.2.1.

Selectivity estimation for strings have been studied in literature: 1-dimensional string estimation [KVI96] and its evaluation [JNS99], multi-dimensional substring estimation [JKN99], etc. In general, selectivity estimation for XML query is more complicated due to the fact that in a tree-shaped query, one needs to take the correlation between paths of the query into consideration.

Therefore, an array of specially-designed selectivity estimation methods are developed recently. For instance, selectivity estimation for path expressions, more restricted than twigs, has been investigated in Lore [MW99] and Niagara [AAN01] projects. More recently, the state-of-the-art technique for tree-shaped query case was proposed in [CJK01]. It proposes an estimation method that, given a twig query, first creates a set of query twiglets and estimates the number of matches of each query twiglet using set hashing and combines the query twiglet estimates into an estimate for the twig query using maximal overlap. Our work directly improves upon it by extending the framework to support selectivity estimation for relaxed twigs case.

Other selectivity estimation methods in query relaxation context (e.g., selectivity for the relaxed queries based on the order as proposed in [KS01]) are also interesting direction to pursue.

5.3 Problem Definition

We consider a problem of selectivity estimation in the context of query relaxation in XML model. Let us denote a set of relaxed twig queries of a twig query Q as $R_Q = \{R_i \mid R_i \text{ is a relaxed twig of } Q\}$. Then, formally, we consider the following problem:

Given a small summary data structure[3] T' corresponding to an XML data tree T, and a twig query Q and a set of Q's relaxed twig queries R_Q, estimate the total number N of the combined twig matches of R_Q, i.e., $N = sel(\sum_{R_i \in R_Q} R_i)$.

One can give a very loose lower and upper bounds of N as follows:

$$MAX_{R_i \in R_Q}\{sel(R_i)\} \leq \ N \ \leq \sum_{R_i \in R_Q} sel(R_i) \tag{5.1}$$

That is, when a particular relaxed query R_i contains the rest of the relaxed queries R_j ($i \neq j$, $R_i, R_j \in R_Q$), N becomes $sel(R_i)$ at its lower bound. On the other hand, if all the relaxed queries in R_Q are disjoint each other without any overlap, then N becomes $\sum_{R_i \in R_Q} sel(R_i)$ at its upper bound. Therefore, our goal in this study is to be able to get a good estimation when neither of these conditions is satisfied (i.e., when there are some overlaps among relaxed queries).

5.4 Method

To solve the problem, two sub-issues are needed to be addressed:

1. *Computing* $sel(R_i)$: Conventional selectivity estimation techniques (e.g., [CJK01, AAN01]) work well to compute $sel(Q)$. We need to find out if this can be readily applicable to compute $sel(R_i)$, where R_i is a relaxed query of Q.

2. *Computing* $sel(\sum_{R_i \in R_Q} R_i)$: Given a set of relaxed queries R_1, ..., R_n and their corresponding selectivity estimates, i.e., $sel(R_1)$, ..., $sel(R_n)$, how to combine them to get the total number of answers N without counting the same answer multiple times?

[3]For instance, CST in [CJK01] or Markov table in [AAN01].

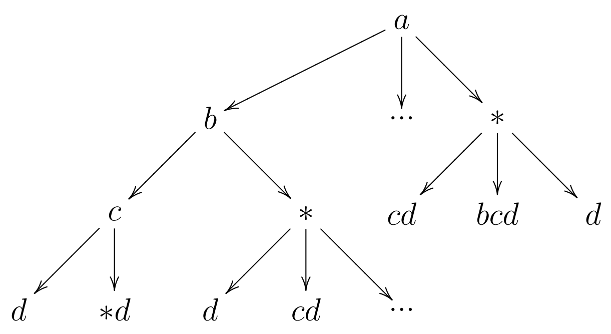


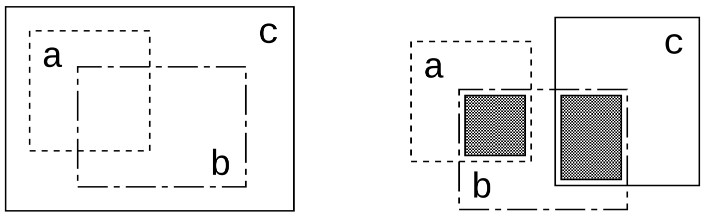
Figure 5.2: A portion of CST having the prefix a.

Our proposal here is inspired by the approach proposed in [CJK01]. Therefore, as a summary data structure, we use the CST, an augmented pruned suffix trie that represents frequency information of small twigs in the data tree. Details of the CST are omitted in the interest of space and interested readers are referred to [CJK01].

5.4.1 Computing $sel(R_i)$

To correctly estimate the selectivity of relaxed twig queries, one must support the case of estimating selectivity of twigs with wildcards ("*"). Toward this end, for each root-to-leaf paths in the data tree, all *-extended paths are generated and inserted into the CST. For instance, for the path $a.b.c.d$, new paths $a.{*}.b.c.d$, $a.b.{*}.c.d$, $a.b.c.{*}.d$, $a.{*}.c.d$, $a.b.{*}.d$, and $a.{*}.d$ are inserted. In addition, all their suffixes are inserted. To keep the CST space manageable, however, we allow at most one "*" in the *-extended paths in the CST A portion of the CST for this scenario is illustrated in Figure 5.2.

Note that relaxed twigs from the original twig query would typically contain "*" symbol when the node deletion or edge relaxation occurred. Then, selectivity estimates of relaxed twigs can be directly obtained using maximal parsing strate-



(a) Fully contained (b) Partially contained

Figure 5.3: Illustration of query containment and overlap.

gies on the twig query paths to include the "*"s appearing in the query. For instance, the selectivity of a relaxed twig $/a//b/c//d$ (i.e., two parent-child edges, a/b and c/d, are relaxed to descendent edges, $a//b$ and $c//d$) can be obtained by locating the path $a.*.b.c.*.d$ from the CST, which would be in turn split into two paths $a.*.b.c$ and $c.*.d$ according to the maximal overlapping strategy. Finally, $sel(a.*.b.c.*.d) = \frac{sel(a*b.c)\times sel(c*d)}{sel(c)}$ by, for instance, *Pure MO* method in [CJK01].

5.4.2 Computing $sel(\sum_{R_i \in R_Q} R_i)$

Consider two cases of the three relaxed queries R_a, R_b and R_c depicted in Figure 5.3. Three estimates $sel(R_a)$, $sel(R_b)$, and $sel(R_c)$ have been already obtained by the method described in Section 5.4.1. Now, the goal is to compute $sel(R_a + R_b + R_c)$. In (a), matches a or b can be ignored since they are fully contained by c. Thus, $sel(R_a + R_b + R_c) = sel(R_c)$. However, in (b), there is no complete containment relationship among three matches and thus $sel(R_a + R_b + R_c) = sel(R_a) + sel(R_b) + sel(R_c) - sel(\text{overlap})$. Hence, the critical issues are (1) to understand the precise meaning of the overlap and (2) express the overlap in some form that can be understood by a summary structure like the CST.

5.4.2.1 Semantics of the Overlap

The basic set theory suggests that $sel(R_a + R_b + R_c) = sel(R_a) + sel(R_b) + sel(R_c) - sel(R_a \cap R_b) - sel(R_b \cap R_c) - sel(R_c \cap R_a) + sel(R_a \cap R_b \cap R_c)$. This formula holds insofar as nodes involved are of the same degree and on the same domain. However, in our case, due to the various relaxations, a relaxed twig has a different set of (projected) nodes than another one (e.g., $/\underline{a}/b$ and $/a/c//\underline{b}$). To resolve this issue, we introduce the notion of extension-compatibility:

Definition 8 (Extension Compatibility) *Given two sets of projected nodes, P_a and P_b, for relaxed twig queries a and b, if $P_a \subseteq P_b$ or $P_a \supseteq P_b$, then, a and b are said* extension-compatible. *Otherwise,* extension-incompatible. $\square$

Definition 9 (Redundancy) *Given two twig matches x and y, if $x \ll y$ holds, where $\ll$ is a partial order over a cross-product space of values where ϵ (i.e., null) $\leq$ any value, then x is said a* redundant *match of y, but not vice versa. When a match x has no redundant match, x is said* irredundant. $\square$

Lemma 1. *If twigs a and b are extension-incompatible, then there is no redundant matches between a and b.* $\blacksquare$

Proof. According to the definition of extension-incompatible twigs, the projected nodes, P_a and P_b, of twigs a and b can be represented as: $P_a = P_c + X$, and $P_b = P_c + Y$, where $P_c = P_a \cap P_b$. Suppose there is redundancy between a and b. Then, there must be some partial order. However, this is impossible; Against the schema $[P_c, X, Y]$, for instance, a and b will have matches in forms $[P_c, X, \epsilon]$ and $[P_c, \epsilon, Y]$, respectively and there is no partial order between these two matches. Thus, proof by contradiction. (q.e.d)

Definition 10 (Projected Set Operators) *For two projected twig queries a and b, (1) $a \sqcap b$ (projected intersection) is a set of redundant matches in a and b, and (2) $a \sqcup b$ (projected union) is a set of irredundant matches in a or b.* □

Lemma 2. $a \sqcup b = a + b - a \sqcap b$ ■

PROOF. Prove both directions:

$\Rightarrow$ Take x in $a \sqcup b$. Then, x is either in a or b.

1. If x is in a: (1) If x is irredundant, then x cannot be in b by the definition of redundancy. Therefore, x is in $a + b$, and in turn in $a + b - a \sqcap b$. (2) If x has a redundant match y in b, then $a \sqcap b$ would choose y over x. Thus, x will remain in $a + b - a \sqcap b$.

2. If x is in b: symmetric.

$\Leftarrow$ Take x in $a + b - a \sqcap b$. Then, x is either in a or b, but must not be in $a \sqcap b$. The fact that x is not in $a \sqcap b$ implies that x has no redundant match. Therefore, x will certainly survive through $\sqcup$ filtering and will be in $a \sqcup b$. (q.e.d)

COROLLARY 1. If twig queries a and b are extension-incompatible, $a \sqcap b = \emptyset$ and $a \sqcup b = a + b$. (q.e.d)

PROOF. Follows from Lemmas 1 and 2. (q.e.d)

Example 5. Overlap between twigs $/\underline{a}/b$ and $/a/c//\underline{b}$ is always empty since two twigs are extension-incompatible. On the other hand, overlap between $/\underline{a}//d/\underline{b}$ and $/\underline{a}/c//\underline{b}$ can be computed by $/\underline{a}//d/\underline{b} \sqcap /\underline{a}/c//\underline{b}$ and its selectivity can be non-zero if there happens to be a branch in an XML data that satisfies the constraint $/a/c//d/b$. □

$$\begin{tikzcd}
 & & a_1 \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] \arrow[drrr] & & & \\
b_1 \arrow[d] & b_2 \arrow[d] & b_3 \arrow[d] & b_4 & b_5 \arrow[d] & b_6 \arrow[d] \\
c_1 \arrow[d] & c_2 & c_3 \arrow[d] & & d_5 & c_4 \arrow[d] \\
d_1 \arrow[d] & & d_4 & & & c_5 \arrow[d] \\
d_2 \arrow[d] & & & & & d_6 \\
d_3 & & & & &
\end{tikzcd}$$

Figure 5.4: Example of XML data tree D.

Let us check if the semantics that we have described so far is consistent with the intuitive meaning of overlap. Figure 5.4 depicts an example XML tree. Suppose that users ask a simple twig query $Q : /\underline{a}/\underline{b}/\underline{c}/\underline{d}$ with a weight threshold θ. Furthermore, suppose only two relaxed queries satisfy the given θ – $R_1 : /\underline{a}/\underline{b}//\underline{c}$ (i.e., edge between b and c is relaxed and node d is removed) and $R_2 : /\underline{a}/\underline{b}//\underline{d}$ (i.e., node c is removed). Users are interested in quickly finding out "the number N of approximate answers that satisfy θ", that is $N = sel(Q + R_1 + R_2)$.

Corresponding matches when these queries are evaluated against D of Figure 5.4 are shown in Table 5.1. Note that queries R_1 and R_2 are extension-incompatible due to their projected nodes, $\{a, b, c\}$ and $\{a, b, d\}$, resulting in no overlaps in-between. Then, $N = sel(Q) + sel(R_1) + sel(R_2) - sel(Q \sqcap R_1) - sel(Q \sqcap R_2) - sel(R_1 \sqcap R_2) + sel(Q \sqcap R_1 \sqcap R_2) = 2 + 5 + 6 - 2 - 2 - 0 + 0 = 9$. When we closely examine the tree D of Figure 5.4, it is not difficult to see that in fact there are 9 distinct approximate matches for queries Q, R_1, and R_2. Figure 5.5 illustrates this point; Single and double lines represent child and descendant relationships, respectively, and dotted lines represent overlapped match. Note that

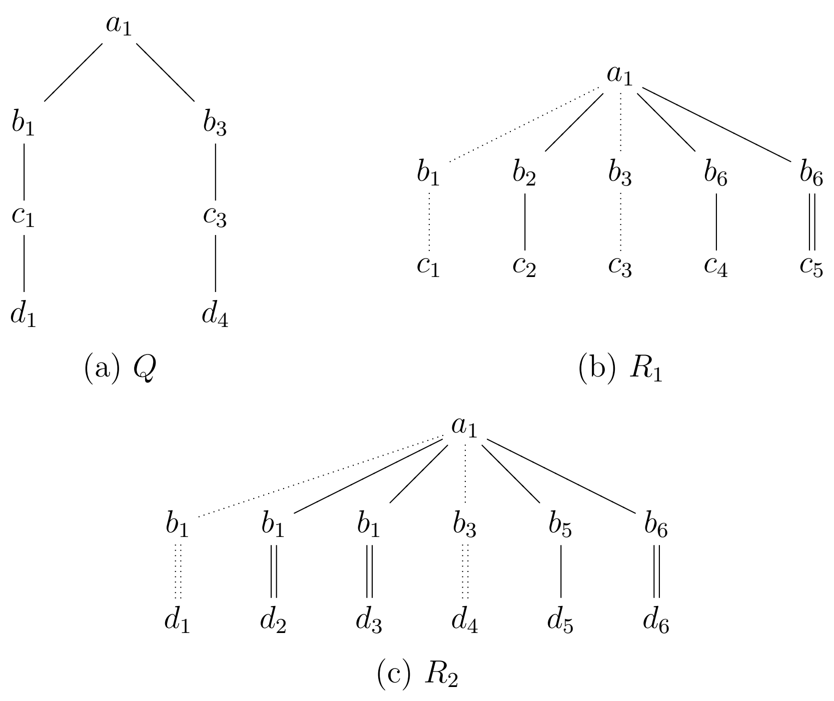


Figure 5.5: Distinct matches of queries Q, R_1, and R_2.

there are a total of 9 solid lines (i.e., matches) in three trees of (a), (b), and (c).

The general formula involving n such relaxed twig queries is as follows:

$$sel(\sum_{i=1}^{n} R_i) = \sum_{i=1}^{\binom{n}{i}} (-1)^{i-1} sel(\sqcap_{j=1}^{i} R_j)$$

In order for the formula to hold under the notion of projected intersection and union, the basic laws such as idempotency, commutativity, associativity, or distributivity must hold. Here, we simply state that they in fact hold and a proof for the case of distributive law is shown below.

Lemma 3. *Distributive law holds for $\sqcap$ and $\sqcup$. For relaxed twigs a, b, and c:*

- $a \sqcup (b \sqcap c) = (a \sqcup b) \sqcap (a \sqcup c)$
- $a \sqcap (b \sqcup c) = (a \sqcap b) \sqcup (a \sqcap c)$ ■

$\$a$	$\$b$	$\$c$	$\$d$
a_1	b_1	c_1	d_1
a_1	b_3	c_3	d_4

(a) $sel(Q) = 2$

$\$a$	$\$b$	$\$c$
a_1	b_1	c_1
a_1	b_2	c_2
a_1	b_3	c_3
a_1	b_6	c_4
a_1	b_6	c_5

(b) $sel(R_1) = 5$

$\$a$	$\$b$	$\$d$
a_1	b_1	d_1
a_1	b_1	d_2
a_1	b_1	d_3
a_1	b_3	d_4
a_1	b_5	d_5
a_1	b_6	d_6

(c) $sel(R_2) = 6$

$\$a$	$\$b$	$\$c$	d
a_1	b_1	c_1	ϵ
a_1	b_3	c_3	ϵ

(d) $sel(Q \sqcap R_1) = 2$

$\$a$	$\$b$	c	$\$d$
a_1	b_1	ϵ	d_1
a_1	b_3	ϵ	d_4

(e) $sel(Q \sqcap R_2) = 2$

$R_1 \sqcap R_2 =$
$Q \sqcap R_1 \sqcap R_2 = \emptyset$

(f) $sel(R_1 \sqcap R_2) =$ $sel(Q \sqcap R_1 \sqcap R_2) = 0$

Table 5.1: Projected twig matches of Q, R_1, R_2 and various overlaps against D of Figure 5.4.

PROOF. Let us prove the former. The latter can be proved similarly.

$\Rightarrow$ Suppose x is in $a \sqcup (b \sqcap c)$. Then, x is either in a or in $(b \sqcap c)$.

- If x is in a, then x is also in $(a \sqcup b)$ as well as $(a \sqcup c)$. Therefore, x is in $(a \sqcup b) \sqcap (a \sqcup c)$.
- If x is in $(b \sqcap c)$, then x is in $(a \sqcup b)$ because x is in b, and x is also in $(a \sqcup c)$, because x is in c. Hence, again x is in $(a \sqcup b) \sqcap (a \sqcup c)$.

$\Leftarrow$ Take x in $(a \sqcup b) \sqcap (a \sqcup c)$. Then x is in $(a \sqcup b)$ as well as in $(a \sqcup c)$.

- If x is in a, then x is trivially in $a \sqcup (b \sqcap c)$.
- If x is in b, then it must also be in c because x is in $(a \sqcup b)$ as well as in $(a \sqcup c)$. Hence, x is in $b \sqcap c$, and therefore it is in $a \sqcup (b \sqcap c)$.
- If x is in c, then it must also be in b similarly. Hence, x is in $b \sqcap c$, and in $a \sqcup (b \sqcap c)$. (q.e.d)

5.4.2.2 Overlap Formula

One remaining problem is that existing summary data structures proposed in [CJK01, AAN01] are unable to handle twigs having the "intersection" operator that are found in $sel(R_i \sqcap R_j)$ in Lemma 2. Therefore, we need to find an alternative twig expression (i.e., overlap formula) that *captures* the twigs involving intersection operator. For instance, observe that any twig matches satisfying the expression $/\underline{a}//d/\underline{b} \sqcap /\underline{a}/c//\underline{b}$ must also satisfy $/\underline{a}/c//d/\underline{b}$, and further $sel(/\underline{a}/c//d/\underline{b})$ can be directly obtained from the CST. Therefore, in this sense, $/\underline{a}/c//d/\underline{b}$ is the overlap formula.

One possible way to find the overlap formula is using the algorithm to compute the intersection of two regular expressions in automata theory. That is, given two relaxed twigs a and b, one can do: (1) convert them into corresponding regular expressions r_a and r_b with sizes n and m, respectively, (2) convert r_a and r_b to non-deterministic finite automata (NFA) whose number of states is linear in n and m, (3) convert the two NFAs to an NFA for the intersection whose size is nm using the property $r_a \cap r_b = \overline{\overline{r_a} \cup \overline{r_b}}$, and (4) finally convert the NFA back to a regular expression, from which the final overlap formula can be inferred easily. However, this method is not suitable for our application since in the last step, the size of the regular expression can blow up (i.e., $O((nm)^3 * 4^{nm})$) and the output regular expression is typically very long and complicated (Chapter 4.3 [HMU01]).

Therefore, instead, we propose a simple rewriting-based algorithm to compose the overlap formula. The meta characteristic of the overlap formula of two relaxed twigs is that the formula would take whatever "tighter" condition of two twigs. Consider two extension-compatible[4] twigs $R = (V_R, E_R, P_R)$ and $S = (V_S, E_S, P_S)$ that are relaxed from a query Q, where common projected nodes are denoted as: $P_O = P_R \cap P_S$. Now, let us describe an algorithm to compute the overlap formula F informally.

1. Imagine R and S as a list of alphabets (ignoring edges / or // for now). Then, comparing R and S can be viewed as follows:

   (a) $R = \alpha, S = \alpha\beta$ (e.g., $R = /a/b, S = /a/b/c$, where $\alpha = ab, \beta = c$)

   (b) $R = \alpha\beta_1, S = \alpha\beta_2$ (e.g., $R = /a/b//c, S = /a/b//d$, where $\alpha = ab, \beta_1 = c, \beta_2 = d$)

   (c) $R = \alpha\gamma, S = \alpha\beta\gamma$ (e.g., $R = /a//c, S = /a/b//c$, where $\alpha = a, \beta = b, \gamma = c$)

[4]If two are extension-incompatible, then the overlap formula would be simply $F = \emptyset$.

(d) $R = \alpha\beta_1\gamma, S = \alpha\beta_2\gamma$ (e.g., $R = /a/b//c, S = /a//d/c$, where $\alpha = a, \beta_1 = b, \beta_2 = d, \gamma = c$)

2. Now, let us denote an edge / or // that precedes an alphabet α in a query R as e^R_α. Further, between two such edges proceeding the *same alphabet*, let us denote the more restricting edge as $\mathbf{e}_\alpha$ (i.e., between edges / and //, the more restricting edge $\mathbf{e}_\alpha$ is /). Then, for each case, the overlap formula F can be stated as follows:

   (a) If $R = e^R_\alpha\alpha, S = e^S_\alpha\alpha e^S_\beta\beta$, then $F = \mathbf{e}_\alpha\alpha e^S_\beta\beta$

   (b) If $R = e^R_\alpha\alpha e^R_\beta\beta_1, S = e^S_\alpha\alpha e^S_\beta\beta_2$, then

$$F = \begin{cases} \mathbf{e}_\alpha\alpha & \text{if } e^R_\beta = e^S_\beta = / \\ \mathbf{e}_\alpha\alpha/\beta_1//\beta_2 & \text{if } e^R_\beta = /, e^S_\beta = // \\ \mathbf{e}_\alpha\alpha/\beta_2//\beta_1 & \text{if } e^R_\beta = //, e^S_\beta = / \\ \mathbf{e}_\alpha\alpha//\beta_1//\beta_2 \sqcup \mathbf{e}_\alpha\alpha//\beta_2//\beta_1 & \text{if } e^R_\beta = e^S_\beta = // \end{cases}$$

   (c) If $R = e^R_\alpha\alpha e^R_\gamma\gamma, S = e^S_\alpha\alpha e^S_\beta\beta e^S_\gamma\gamma$, then

$$F = \begin{cases} \emptyset & \text{if } e^R_\gamma = e^S_\beta = / \\ \mathbf{e}_\alpha\alpha/\gamma//\beta e^S_\gamma\gamma & \text{if } e^R_\gamma = /, e^S_\beta = // \\ \mathbf{e}_\alpha\alpha e^S_\beta\beta e^S_\gamma\gamma & \text{if } e^R_\gamma = //, e^S_\beta = / \\ S & \text{if } e^R_\gamma = e^S_\beta = // \end{cases}$$

   (d) If $R = e^R_\alpha\alpha e^R_\beta\beta_1 e^R_\gamma\gamma, S = e^S_\alpha\alpha e^S_\beta\beta_2 e^S_\gamma\gamma$, then

$$F = \begin{cases} \mathbf{e}_\alpha\alpha//\mathbf{e}_\gamma\gamma & \text{if } e^R_\beta = e^S_\beta \\ \mathbf{e}_\alpha\alpha/\beta_1//\beta_2\mathbf{e}_\gamma\gamma & \text{if } e^R_\beta = /, e^S_\beta = // \\ \mathbf{e}_\alpha\alpha/\beta_2//\beta_1\mathbf{e}_\gamma\gamma & \text{if } e^R_\beta = //, e^S_\beta = / \end{cases}$$

3. Mark remaining alphabets that belong to P_O as projected nodes of F. Repeatedly apply above two steps into blocks of two queries.

For instance, imagine various forms of relaxed queries from a query Q : $/\underline{a}/\underline{b}/\underline{c}/\underline{d}$.

- Step 2.(a): $/\underline{a}//\underline{b} \sqcap /\underline{a}/\underline{b}//\underline{d} \equiv /\underline{a}/\underline{b}//d$
- Step 2.(b): $/\underline{a}/\underline{b}/c \sqcap /\underline{a}//\underline{b}//d \equiv /\underline{a}/\underline{b}/c//d$
- Step 2.(c): $/\underline{a}/\underline{c} \sqcap /\underline{a}/\underline{b}//\underline{c} \equiv \emptyset$
- Step 2.(d): $/\underline{a}//b/\underline{c} \sqcap /\underline{a}/d//\underline{c} \equiv /\underline{a}/d//b/\underline{c}$

More complex cases can be similarly computed by somewhat like a divide-and-conquer manner. For instance, $/\underline{a}/\underline{b}//\underline{c}/\underline{d} \sqcap /\underline{a}//\underline{c}$ can be treated as if $R/\underline{d} \sqcap S$, where $R = /\underline{a}/\underline{b}//\underline{c}$ and $S = /\underline{a}//\underline{c}$. Computing $R \sqcap S$ is the case of 2.(c) and thus results in $R \sqcap S \equiv O$, where $O = /\underline{a}/b//\underline{c}$. Then, computing $O/\underline{d} \sqcap O$ is the case of 2.(a), resulting in $O/\underline{d} \sqcap O = O/d$. Therefore, the final query that captures the given projected intersection is $/\underline{a}/b//\underline{c}/d$.

By using the described algorithm, one can find a formula that captures the overlap of two projected queries. The only problem in the algorithm is the forth case of 2.(b) which results in a formula with the projected union operator. For instance, the overlap formula of two relaxed twigs $/\underline{a}//\underline{b}/\underline{c}/\underline{d}$ and $/\underline{a}/\underline{b}//\underline{d}$ is $/\underline{a}/\underline{b}/\underline{c}/\underline{d} \sqcup /\underline{a}/\underline{b}//\underline{b}/\underline{c}/\underline{d}$. This case is problematic since like the projected intersection operator, the projected union operator cannot be handled by the summary data structures. If this occurs, we assume that the selectivity of the overlap between two relaxed twigs be the maximum of the selectivities of two twigs: $sel(/\underline{a}//\underline{b}/\underline{c}/\underline{d} \sqcap /\underline{a}/\underline{b}//\underline{d}) = MAX\{sel(/\underline{a}/\underline{b}/\underline{c}/\underline{d}), sel(/\underline{a}/\underline{b}//\underline{b}/\underline{c}/\underline{d})\}$.

Finally, let us go through a relatively complex example having *relaxed twigs* to illustrate what we have discussed so far. Consider a data tree in Figure 5.6 and queries in Figure 5.7. Assume the ordered model. Both relaxed queries R_1

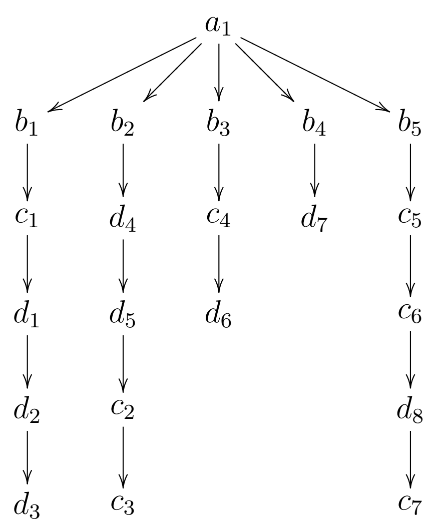


Figure 5.6: Example of XML data tree E.

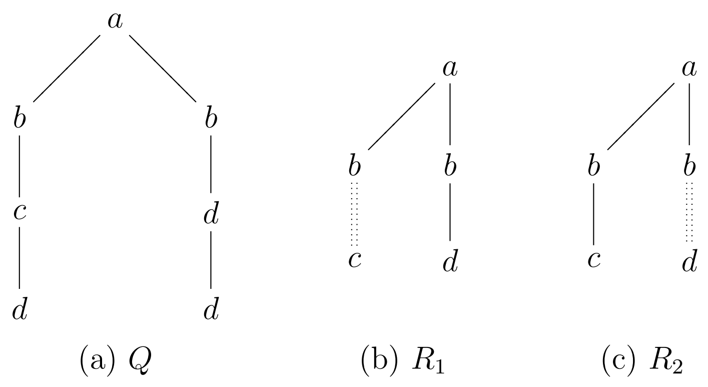


Figure 5.7: Original query Q and its two relaxed queries R_1 and R_2.

and R_2 have two relaxations occurred from the original query Q. This example is peculiar since multiple bindings of a node is possible. That is, the node d in the right branch of R_1 can be bound to any of the two ds in the right branch of Q. This also applies to the case of R_2. To differentiate the different bindings of the node d, let us assume the following answer schema for the original query Q: $(A, B_1, C, D_1, B_2, D_2, D_3)$, where the columns 1,2,3,4 correspond to the left branch and the column 1,5,6,7 correspond to the right branch of Q.

First, in the total semantics, each answer set is shown below:

$$
\begin{aligned}
Q &= \{(a_1, b_1, c_1, d_1, b_2, d_4, d_5), (a_1, b_3, c_4, d_6, b_2, d_4, d_5)\} \\
R_1 &= \{(a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon), (a_1, b_1, c_1, \epsilon, b_4, d_7, \epsilon), (a_1, b_2, c_2, \epsilon, b_2, d_4, \epsilon), \\
&\quad (a_1, b_2, c_2, \epsilon, b_4, d_7, \epsilon), (a_1, b_2, c_3, \epsilon, b_2, d_4, \epsilon), (a_1, b_2, c_3, \epsilon, b_4, d_7, \epsilon), \\
&\quad (a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon), (a_1, b_3, c_4, \epsilon, b_4, d_7, \epsilon), (a_1, b_5, c_5, \epsilon, b_2, d_4, \epsilon), \\
&\quad (a_1, b_5, c_5, \epsilon, b_4, d_7, \epsilon), (a_1, b_5, c_6, \epsilon, b_2, d_4, \epsilon), (a_1, b_5, c_6, \epsilon, b_4, d_7, \epsilon), \\
&\quad (a_1, b_5, c_7, \epsilon, b_2, d_4, \epsilon), (a_1, b_5, c_7, \epsilon, b_4, d_7, \epsilon)\} \\
R_2 &= \{(a_1, b_1, c_1, \epsilon, b_1, d_1, \epsilon), (a_1, b_1, c_1, \epsilon, b_1, d_2, \epsilon), (a_1, b_1, c_1, \epsilon, b_1, d_3, \epsilon), \\
&\quad (a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon), (a_1, b_1, c_1, \epsilon, b_2, d_5, \epsilon), (a_1, b_1, c_1, \epsilon, b_3, d_6, \epsilon), \\
&\quad (a_1, b_1, c_1, \epsilon, b_4, d_7, \epsilon), (a_1, b_1, c_1, \epsilon, b_5, d_8, \epsilon), (a_1, b_3, c_4, \epsilon, b_1, d_1, \epsilon), \\
&\quad (a_1, b_3, c_4, \epsilon, b_1, d_2, \epsilon), (a_1, b_3, c_4, \epsilon, b_1, d_3, \epsilon), (a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon), \\
&\quad (a_1, b_3, c_4, \epsilon, b_2, d_5, \epsilon), (a_1, b_3, c_4, \epsilon, b_3, d_6, \epsilon), (a_1, b_3, c_4, \epsilon, b_4, d_7, \epsilon), \\
&\quad (a_1, b_3, c_4, \epsilon, b_5, d_8, \epsilon), (a_1, b_5, c_5, \epsilon, b_1, d_1, \epsilon), (a_1, b_5, c_5, \epsilon, b_1, d_2, \epsilon), \\
&\quad (a_1, b_5, c_5, \epsilon, b_1, d_3, \epsilon), (a_1, b_5, c_5, \epsilon, b_2, d_4, \epsilon), (a_1, b_5, c_5, \epsilon, b_2, d_5, \epsilon), \\
&\quad (a_1, b_5, c_5, \epsilon, b_3, d_6, \epsilon), (a_1, b_5, c_5, \epsilon, b_4, d_7, \epsilon), (a_1, b_5, c_5, \epsilon, b_5, d_8, \epsilon)\} \\
Q \sqcap R_1 &= \{(a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon), (a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon)\} \\
Q \sqcap R_2 &= \{(a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon), (a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon)\} \\
R_1 \sqcap R_2 &= \{(a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon), (a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon) \\
&\quad (a_1, b_5, c_5, \epsilon, b_2, d_4, \epsilon)(a_1, b_1, c_1, \epsilon, b_4, d_7, \epsilon), \\
&\quad (a_1, b_3, c_4, \epsilon, b_4, d_7, \epsilon)(a_1, b_5, c_5, \epsilon, b_4, d_7, \epsilon)\}
\end{aligned}
$$

$$Q \sqcap R_1 \sqcap R_2 = \{(a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon), (a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon)\}$$

Therefore, $sel(Q + R_1 + R_2) = sel(Q) + sel(R_1) + sel(R_2) - sel(Q \sqcap R_1) - sel(Q \sqcap R_2) - sel(R_1 \sqcap R_2) + sel(Q \sqcap R_1 \sqcap R_2) = 2 + 14 + 24 - 2 - 2 - 6 + 2 = 32$. One can easily verify that there are in fact only 32 *irredundant* answers of Q,

Shape	DBLP	SPROT
	NumBranch n, Height h	
PATH	$n = 1$, $2 \leq h \leq 4$	$n = 1$, $2 \leq h \leq 6$
BS	$3 \leq n \leq 5$, $2 \leq h \leq 3$	$3 \leq n \leq 5$, $2 \leq h \leq 4$
DS	N/A	$1 \leq n \leq 3$, $4 \leq h \leq 6$
BAL	$1 \leq n \leq 5$, $2 \leq h \leq 4$	$1 \leq n \leq 5$, $2 \leq h \leq 6$

Table 5.2: Different shapes of query sets.

Type	DBLP	SPROT
	NumRlxQry q, NumRlx r	
A	$1 \leq q \leq 3$, $1 \leq r \leq 2$	$1 \leq q \leq 3$, $1 \leq r \leq 2$
B	$1 \leq q \leq 3$, $2 \leq r \leq 3$	$1 \leq q \leq 3$, $2 \leq r \leq 5$
C	$3 \leq q \leq 5$, $1 \leq r \leq 2$	$3 \leq q \leq 5$, $1 \leq r \leq 2$
D	$3 \leq q \leq 5$, $2 \leq r \leq 3$	$3 \leq q \leq 5$, $2 \leq r \leq 5$

Table 5.3: Different degrees of relaxations.

R_1 and R_2. That is, both R_1 and R_2 have the following two redundant answer sets: $\{(a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon), (a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon)\}$, and $\{(a_1, b_1, c_1, \epsilon, b_2, d_4, \epsilon),$ $(a_1, b_1, c_1, \epsilon, b_4, d_7, \epsilon)$, $(a_1, b_3, c_4, \epsilon, b_2, d_4, \epsilon)$, $(a_1, b_3, c_4, \epsilon, b_4, d_7, \epsilon)$, $(a_1, b_5, c_5, \epsilon, b_2, d_4, \epsilon)$, $(a_1, b_5, c_5, \epsilon, b_4, d_7, \epsilon)\}$, respectively. Therefore, the correct number of irredundant selectivity would be: $sel(Q) + sel(R_1) + sel(R_2) - sel(\text{redundancy}) = 2 + 14 + 24 - 6 = 32$, which is precisely identical to our calculation.

5.5 Experimental Results

5.5.1 Experimental Setup

Data set: We used two real XML data sets in our experiments.

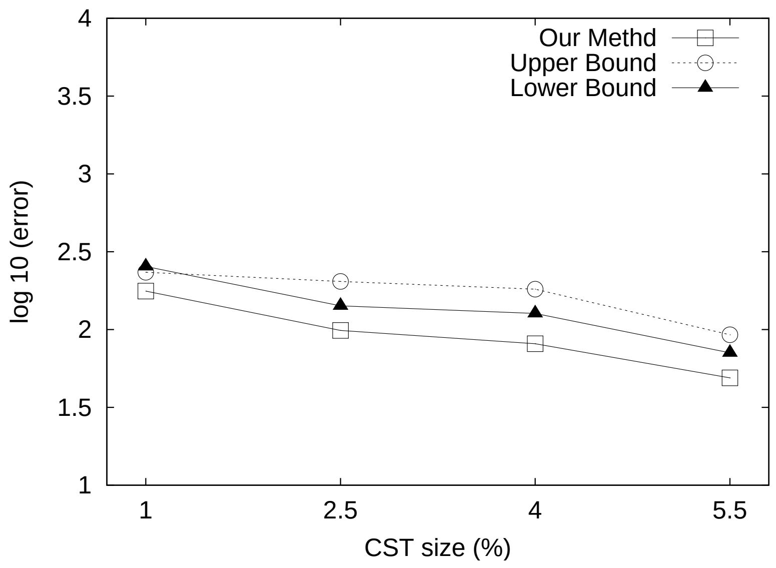


Figure 5.8: Error as the CST space increases (Dataset=SPROT, Shape=BAL, Relationship=CHILD, Type=A).

- DBLP[5]: The data size is about 10MB, and consists of a forest with children such as `book` and `incollection` that in turn have a variable number of children such as `author`, `publisher`, etc. This data set is relatively *flat* (i.e., most instances have at most depth 2 or 3) and *regular* (i.e., the types of sub-elements that every root nodes have are very similar).

- SPROT[6]: This contains annotated protein sequences, including the sequences, annotations, authors, places, citations, etc. The size that we tested is about 5MB, but its schema is far more complex than the DBLP dat set. Its structure is rather *irregular* in that the types of sub-elements under the same element vary greatly.

Algorithm: We implemented three methods (i.e., lowerbound, upperbound, and

[5]ftp://ftp.informatik.uni-trier.de/pub/users/Ley/bib/records.tar.gz
[6]http://www.expasy.ch/sprot

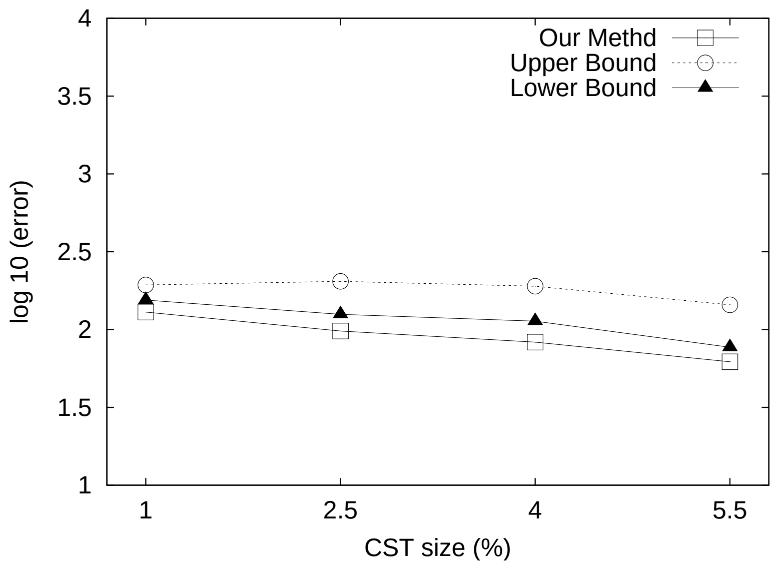


Figure 5.9: Error as the CST space increases (Dataset=SPROT, Shape=BAL, Relationship=CHILD, Type=B).

our proposed methods) in Equation 5.1 of Section 5.3. Our implementation is the extension of the MSH algorithm (selectivity estimation method for twigs) and CST summary structure (correlated subpath tree) in [CJK01], which was shown the best among the proposed methods.

Query set: We experimented with various forms of query sets with 1,000 queries each in it using two factors: NumBranch (# of branches in a twig) and Height (# of nodes in the path). By combination, we generated four different shapes of query sets – PATH (trivial path), BS (bushy & shallow), DS (deep & skinny) and BAL (balanced) – shown in Table 5.2. Since the DBLP data set is rather flat, we did not tested the DS shape.

Another factor that affects to the query shapes is that of how relaxation occurs. For this purpose, we manipulated two parameters: NumRlxQry (# of relaxed queries per original query) and NumRlx (# of relaxations occurred for

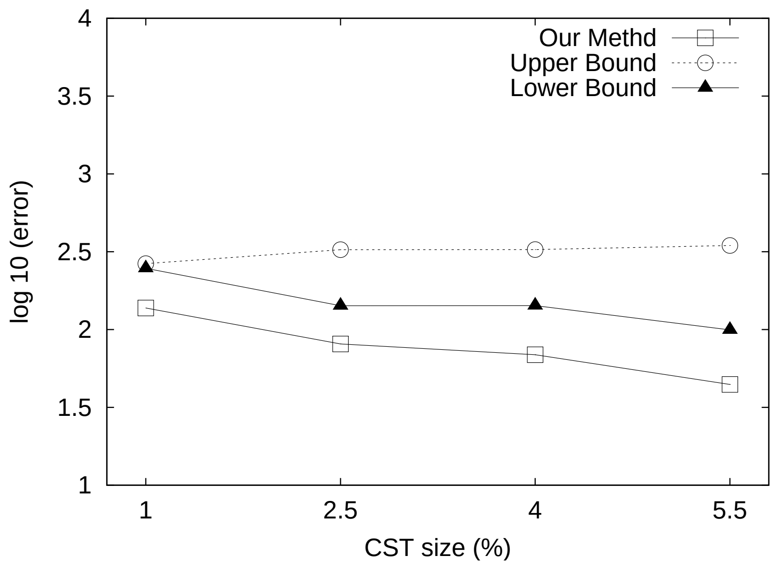


Figure 5.10: Error as the CST space increases (Dataset=SPROT, Shape=BAL, Relationship=CHILD, Type=C).

each relaxed query). By combination, we again generated four sets – A, B, C, D – shown in Table 5.3. These four types roughly correspond to the degrees of relaxations with A being the lightest relaxations and D being the heaviest relaxations.

Finally, we differentiated two sets of queries – one with only parent-child relationship, and the other with both parent-child and ancestor-descendant relationships and denote them by CHILD and BOTH, respectively.

In total, we have generated query sets: $|\text{data set}| \times |\text{shape}| \times |\text{type}| \times |\text{relationship}| = 2 \times 4 \times 4 \times 2 = 64$. In the interest of the space, therefore, we will discuss only representative results for the rest of the Chapter.

Error metric: Following [CJK01], we used *Average Relative Squared Error*,

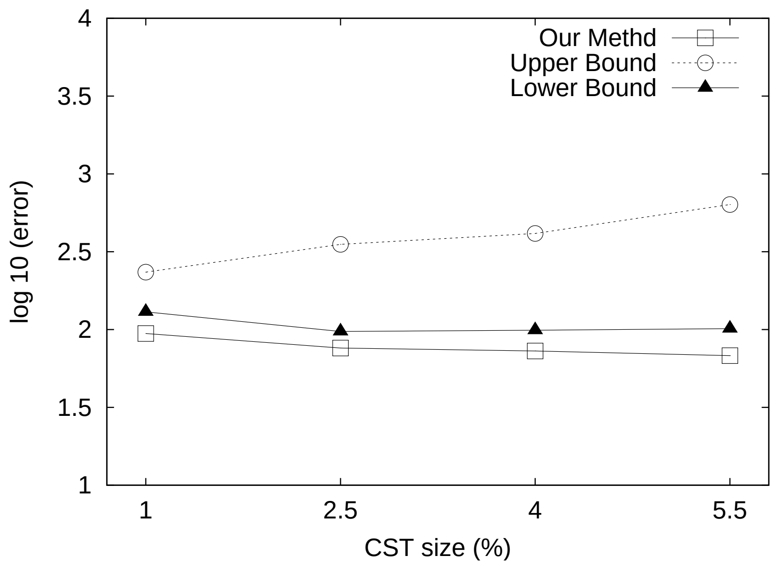


Figure 5.11: Error as the CST space increases (Dataset=SPROT, Shape=BAL, Relationship=CHILD, Type=D).

defined as:

$$Error = \frac{1}{|Q|}\sum_{i\in Q}\frac{(Q_i - Q_i')^2}{Q_i'} \tag{5.2}$$

where Q is a workload of test queries, Q_i is the true selectivity to a query and Q_i' is our estimate.

5.5.2 Accuracy and Time

Figures 5.8, 5.9, 5.10, and 5.11 show the average relative squared error of all methods for the SPROT data set and the BAL query set as the CST size increases. The results for the DBLP data set exhibit similar patterns (except it requires less space than the SPROT due to its simpler structure) and omitted. As the space increases, the performance of all three methods improves steadily regardless of the types of relaxations. Further, one can notice that the patterns between the

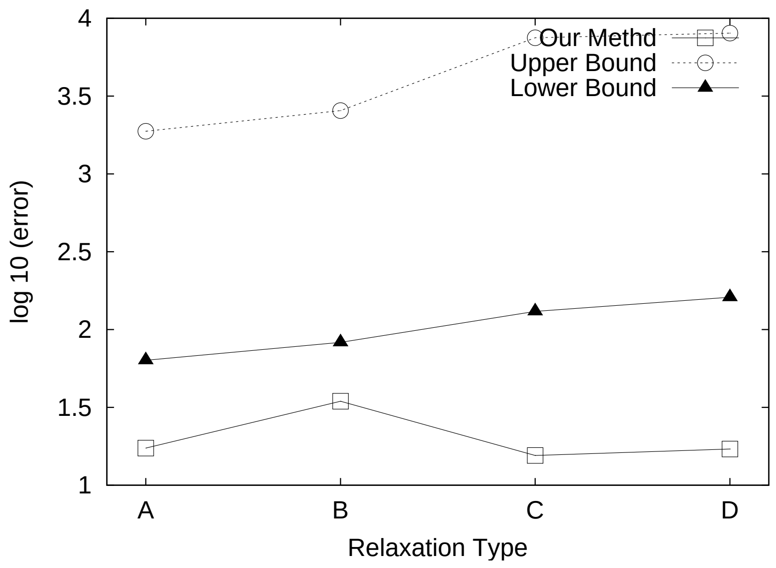


Figure 5.12: Error as the relaxation types change (Dataset=DBLP, Relationship=CHILD, Shape=PATH).

relaxation types A and B and the relaxation types C and D are more similar, suggesting the NumRlxQry parameter plays more major role than the NumRlx parameter.

Figures 5.12 and 5.13 show the average relative squared error of the three methods for the DBLP data set as the relaxation types change. It again has the similar trends as the case for the SPROT data set in Figures 5.8, 5.9, 5.10, and 5.11; that is, our proposal outperforms the other two, where the lowerbound method is the second. For the most experiments that we have conducted for the DBLP set, we found that lowerbound method typically closely follows the curve of our proposal while upperbound method is rather far apart. This can be explained as follows. Since the DBLP data set has a relatively shallow and flat schema, even the relaxed queries could not find much more new answers. For instance, Figure 5.14 shows a typical original query and its two relaxed queries

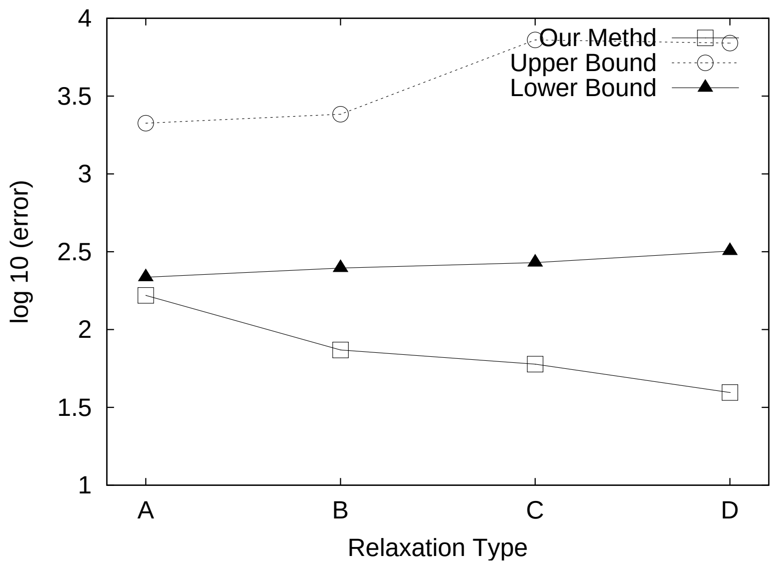


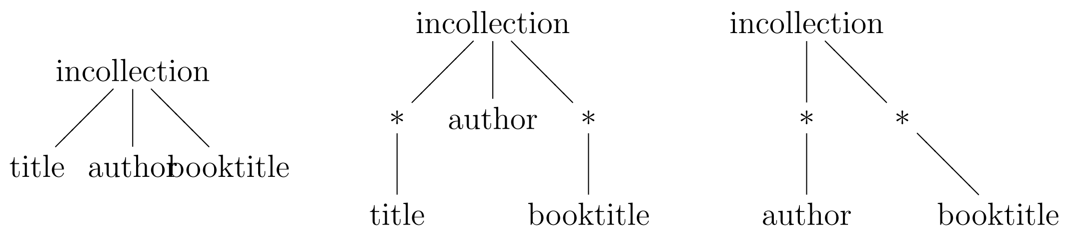
Figure 5.13: Error as the relaxation types change (Dataset=DBLP, Relationship=CHILD, Shape=BAL).

during the actual run of the DBLP test set, which has the following schema:

```
<!ELEMENT dblp (article|incollection|...)*>
<!ELEMENT incollection (author|title|...)*>
```

Note that there are no recursive elements. Further, between `incollection` and its sub-elements, no other sub-elements are permitted. Therefore, for instance, both queries `//incollection/author` and `//incollection//author` will most likely have the same selectivity for the DBLP data set. Therefore, $sel(Q+R_1+R_2)$ is essentially the same as $sel(Q)$ once redundant answers are pruned. In such a case when overlaps among queries constitute a major portion of the answer space, lowerbound method does comparably a good job.

Figures 5.15 and 5.16 shows the different percentages of queries in terms of the distribution of error ratios among three methods. In Figure 5.15, the absolute error ratio for our proposed method is the best among three methods and



(a) Q (b) R_1 (c) R_2

Figure 5.14: Original query Q and its two relaxed queries R_1 and R_2 for the DBLP data set.

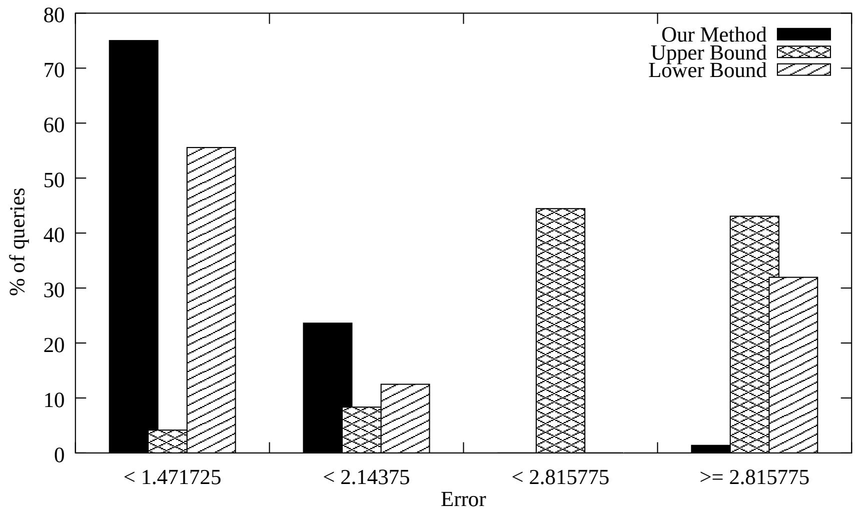


Figure 5.15: Percentage of queries for the absolute error (Dataset=SPROT, Shape=DS, Relationship=CHILD, Type=D).

is the direct reason why our proposal outperforms the other two methods. More importantly, as shown in Figure 5.16, upperbound method tends to over-estimate and lowerbound method tends to under-estimate. This is no surprise since they do not consider "overlap" in their estimation. As the original query tends to have more number of relaxed queries ($3 \leq$NumRlxQry≤ 5) and more number of relaxations occurred in each case ($2 \leq$NumRlx≤ 5), new relaxed queries will inevidently contain more new answers, creating more disjoint answer space. However, the fact that the degree of over-estimation of the upperbound method is

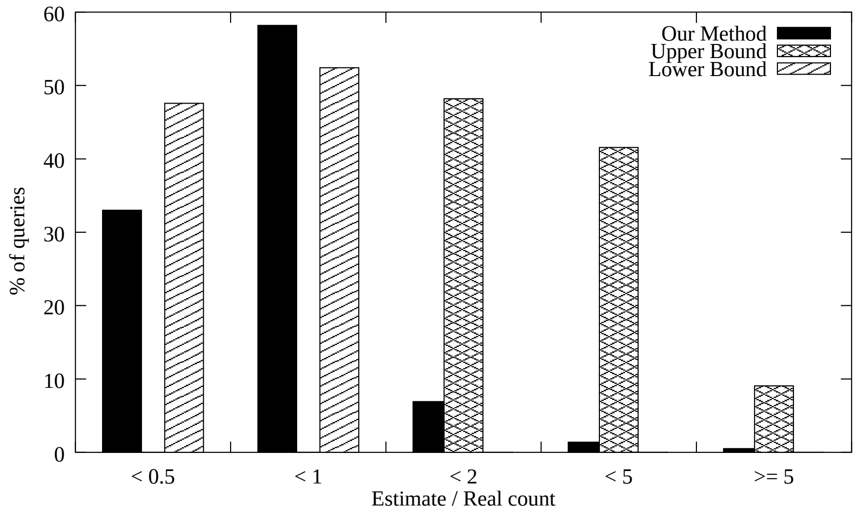


Figure 5.16: Percentage of queries for the relative error (Dataset=SPROT, Shape=DS, Relationship=CHILD, Type=D).

far severe than that of the under-estimation of lowerbound method suggests that this particular query set (Dataset=SPROT, Shape=DS, Relationship=CHILD, Type=D) has less number of "disjoint" answers than "overlapped" answers.

Figures 5.17, 5.18, 5.19, and 5.20 show the average relative squared error of the three methods for the query sets with different relationships against the SPROT data set. Note that the patterns remain largely intact between two query sets with the CHILD and BOTH relationships. This phenomenon can be also witnessed in query sets with different configurations. As a conclusion, our proposal consistently delivers a good estimate regardless of the factors affecting the query characteristics.

All our experiments were performed on a Sun Sparc Ultra-4 machine with 256MB of memory. It takes about 15 minutes to construct and prune the CSTs for all cases and data sets. Estimations for 1,000 queries in each set take about 2 minutes. In all, construction and estimation are fast.

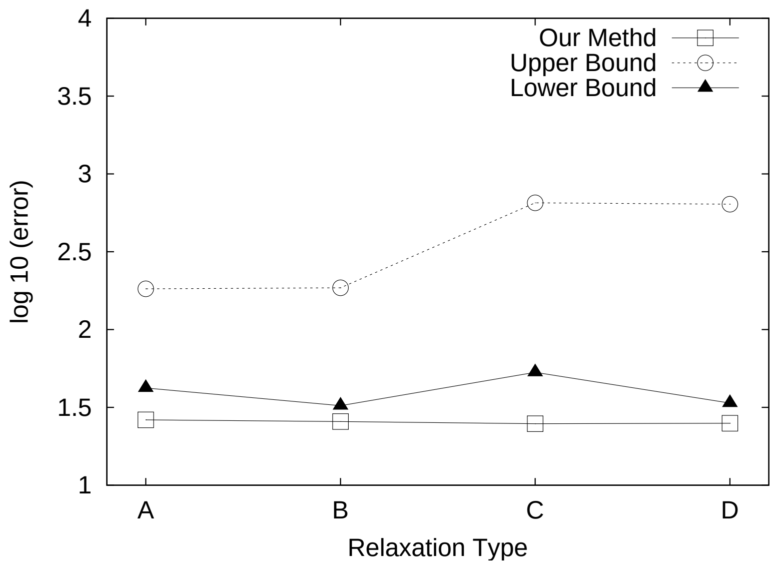


Figure 5.17: Error with different *relationships* as the relaxation types change (Dataset=SPROT, Shape=PATH, Relationship=CHILD).

5.6 Summary

In this Chapter, we look at the problem of estimating the number of answers of an XML twig query, when certain query relaxations are permitted. We show that this problem can be effectively addressed by enhancing the techniques used for estimating the selectivity of the unrelaxed query, in two ways.

- Augment the CST to include paths with the '*' wildcard character, to permit *individual* relaxed queries to be accurately estimated.
- Estimate the overlaps between different relaxed queries, by approximating this overlap by a relaxed query itself; often, this approximation is lossless. This permits the *union* of individual relaxed queries to be accurately estimated.

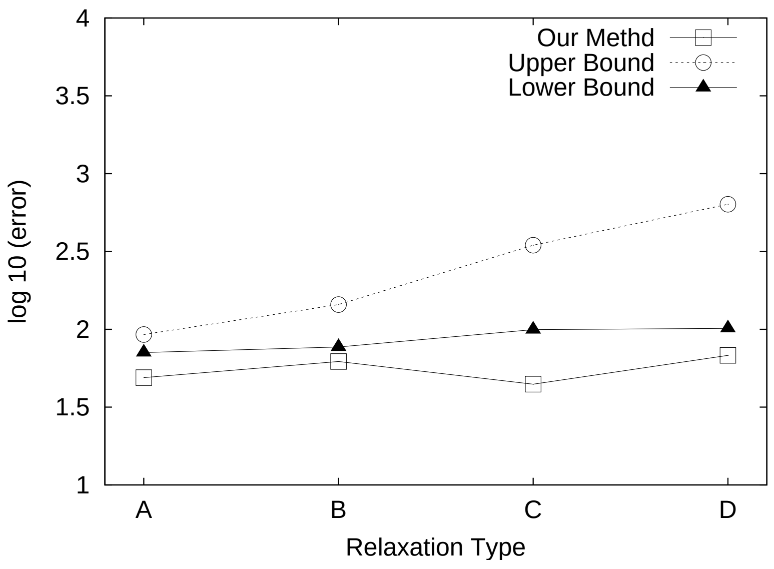


Figure 5.18: Error with different *relationships* as the relaxation types change (Dataset=SPROT, Shape=PATH, Relationship=CHILD).

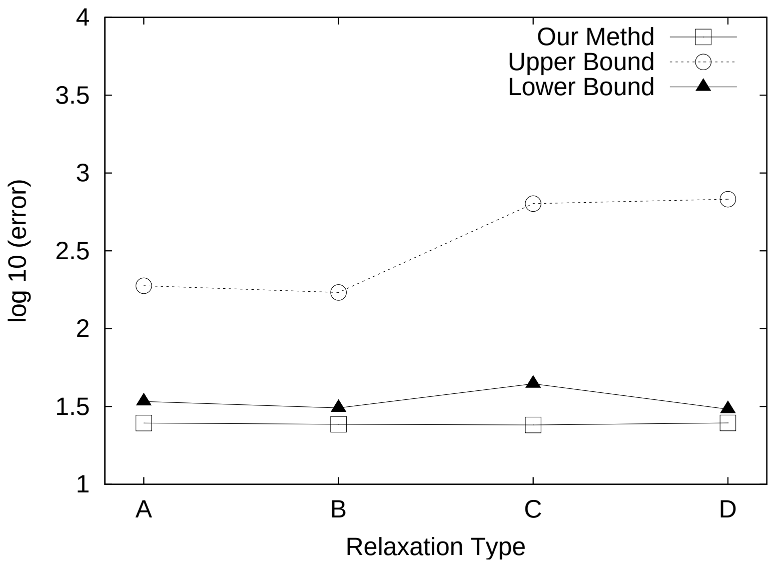


Figure 5.19: Error with different *relationships* as the relaxation types change (Dataset=SPROT, Shape=PATH, Relationship=BOTH).

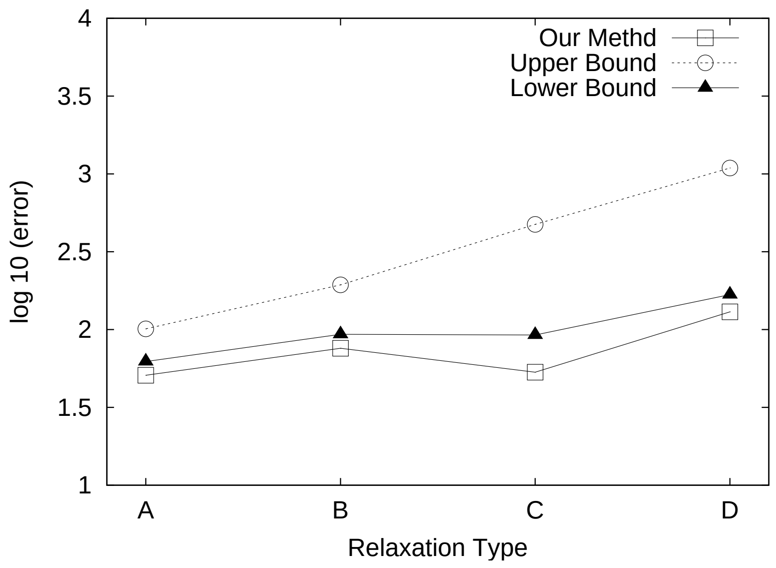


Figure 5.20: Error with different *relationships* as the relaxation types change (Dataset=SPROT, Shape=BAL, Relationship=BOTH).

We experimentally demonstrate that our techniques are much better than competing techniques using the real DBLP and SPROT datasets.

Our work is the first to explore the problem of selectivity estimation of tree pattern relaxations, and opens up many interesting directions of future work. How does one optimize the evaluation of relaxed twig pattern queries taking our estimates into account? How can one quickly identify the dominant (i.e., the ones contributing most of the answers) relaxed queries? We have established the foundations for a lot of interesting work in the area of approximate XML query processing.

CHAPTER 6

XML to Relational Conversion

As XML is emerging as the data format of the Internet era, there are increasing needs to efficiently store and query XML data. One path to this goal is transforming XML data into relational format in order to use relational database technology. Although several transformation algorithms exist, they are incomplete in the sense that they focus only on *structural* aspects and ignore *semantic* aspects. In this Chapter, we present the semantic knowledge that needs to be captured during transformation to ensure a correct relational schema. Furthermore, we develop an algorithm that can (1) derive such semantic knowledge from a given XML Document Type Definition (DTD) and (2) preserve the knowledge by representing it as *semantic constraints* in relational database terms. By combining existing transformation algorithms and our *constraints-preserving* algorithm, one can transform XML DTD to relational schema where correct semantics and behaviors are guaranteed by the preserved constraints.

6.1 Background

As the World-Wide Web becomes a major means of disseminating and sharing information, Extensible Markup Language (XML) [BPS00] is emerging as a possible candidate data format because it is simpler than SGML, and more powerful than HTML. One way to query XML data is to reuse the established relational database techniques by converting and storing XML data in relational storage. Since the hierarchical XML and the flat relational data models are not fully compatible, the transformation is not a straightforward task.

To this end, several XML-to-relational transformation algorithms have been proposed [DFS98, FK99, STH99]. For instance, [STH99] presents 3 algorithms that focus on the table level of the schema while [FK99] studies different performance issues among 8 algorithms that focus on the attribute and value level of the schema. They all transform the given XML Document Type Definition (DTD) to relational schema. Similarly, [DFS98] presents a data mining-based algorithm that instead uses XML documents directly without a DTD.

Although they work well for the given applications, they miss one important point. That is, the transformation algorithms only capture the *structure* of a DTD and ignore the hidden *semantic* constraints. Consider the following example.

Example 6. Consider a DTD modeling conference publications:

```
<!ELEMENT conf   (title,society,year,mon?,paper+)>
<!ELEMENT paper  (pid,title,abstract?)>
```

Suppose the combination of `title` and `year` uniquely identifies the `conf`. Using the hybrid inlining algorithm (explained in Section 6.3), the DTD would be transformed to the following relational schema:

```
conf  (title,society,year,mon)
paper (pid,title,conf_title,conf_year,abstract)
```

While the relational schema correctly captures the structural aspect for the DTD, it does not force correct semantics. For instance, it cannot prevent a tuple t_1: `paper(100,'DTD...','ER',3000,'...')` from being inserted. However,

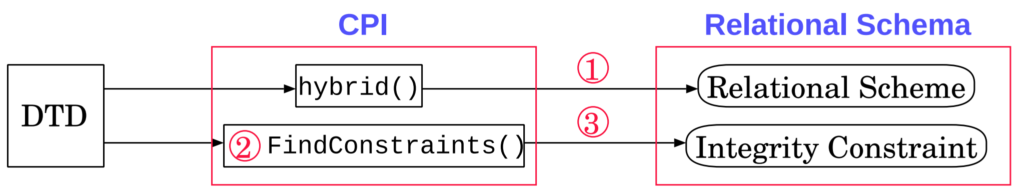


Figure 6.1: Overview of **CPI** algorithm.

tuple t_1 is inconsistent with semantics of the given DTD since the DTD implies that the paper cannot exist without being associated with a conference and there is apparently no conference "ER-3000" yet. In database terms, this kind of violation can be easily prevented by an *inclusion dependency* saying "`paper[conf_title,conf_year]` $\subseteq$ `conf[title,year]`". □

The reason for this inconsistency between the DTD and the transformed relational schema is that transformation algorithms only capture the *structure* of the DTD and ignore the hidden *semantic* constraints. Via our *constraints-preserving inlining (CPI)* algorithm, we show the kinds of semantic constraints that can be derived from DTDs during transformation, and illustrate how to preserve them by rewriting them in an output schema notation. Since our algorithm to capture and preserve semantic constraints from DTDs is independent of the transformation algorithms, our algorithm can be applied to various transformation processes such as [DFS98, FK99, STH99] with little change. Figure 6.1 presents an overview of our approach. First, given a DTD, we transform it to a corresponding relational scheme using an existing algorithm. Second, during the transformation, we discover various semantic constraints in XML notation. Third, we rewrite the discovered constraints to conform to relational notation.

For the rest of the Chapter, we will use the example DTD and XML document in Tables 6.1 and 6.2.

```
<!ELEMENT conf      (title,date,editor?,paper*)>
<!ATTLIST conf      id     ID           #REQUIRED>
<!ELEMENT title     (#PCDATA)>
<!ELEMENT date      EMPTY>
<!ATTLIST date      year   CDATA        #REQUIRED
                    mon    CDATA        #REQUIRED
                    day    CDATA        #IMPLIED>
<!ELEMENT editor    (person*)>
<!ATTLIST editor    eids   IDREFS       #IMPLIED>
<!ELEMENT paper     (title,contact?,author,cite?)>
<!ATTLIST paper     id     ID           #REQUIRED>
<!ELEMENT contact   EMPTY>
<!ATTLIST contact   aid    IDREF        #REQUIRED>
<!ELEMENT author    (person+)>
<!ATTLIST author    id     ID           #REQUIRED>
<!ELEMENT person    (name,(email|phone)?)>
<!ATTLIST person    id     ID           #REQUIRED>
<!ELEMENT name      EMPTY>
<!ATTLIST name      fn     CDATA        #IMPLIED
                    ln     CDATA        #REQUIRED>
<!ELEMENT email     (#PCDATA)>
<!ELEMENT phone     (#PCDATA)>
<!ELEMENT cite      (paper*)>
<!ATTLIST cite      id     ID           #REQUIRED
                    format (ACM|IEEE)   #IMPLIED>
```

Table 6.1: Example of a DTD for `Conference`.

6.2 Related Work

Conversion between different models has been extensively investigated [ZCF97]. For instance, [CAC94] deals with transformation problems in the OODB area; since OODB is a richer environment than RDB, their work is not readily applicable to our application. The logical database design methods and their associated transformation techniques to other data models have been extensively studied in ER research. For instance, [BCN92] presents an overview of such techniques. However, due to the differences between ER and XML models, those transformation techniques need to be modified substantially. More recently, [BHP00] studies a generic mapping between arbitrary models with the focus of developing a framework for model management. Apart from conversion approaches, it is worthwhile to note that there have been also recent investigations on native XML storage systems such as [KM00].

```
<conf id="er05">
  <title>Int'l Conference on Conceptual Modeling (ER)</title>
  <date>
    <year>2005</year> <mon>May</mon> <day>20</day>
  </date>
  <editor eids="sheth bossy">
    <person id="klavans">
      <name fn="Judith" ln="Klavans"/>
      <email>klavans@cs.columbia.edu</email>
    </person>  </editor>
  <paper id="p1">
    <title>Indexing Model for Structured...</title>
    <contact aid="dao"/>
    <author>
      <person id="dao"><name fn="Tuong" ln="Dao"/></person>
    </author>
  </paper>
  <paper id="p2">
    <title>Logical Information Modeling of...</title>
    <contact aid="shah"/>
    <author>
      <person id="shah">
        <name fn="Kshitij" ln="Shah"/>
      </person>
      <person id="sheth">
        <name fn="Amit" ln="Sheth"/>
        <email>amit@cs.uga.edu</email>
      </person>
    </author>
    <cite id="c100" format="ACM">
      <paper id="p3">
        <title>Making Sense of Scientific...</title>
        <author>
          <person id="bossy">
            <name fn="Marcia" ln="Bossy"/>
            <phone>391.4337</phone>
          </person>
        </author> </paper> </cite> </paper>
</conf>
<paper id="p7">
  <title>Constraints-preserving Transformation from...</title>
  <contact aid="lee"/>
  <author>
    <person id="lee">
      <name fn="Dongwon" ln="Lee"/>
      <email>dongwon@cs.ucla.edu</email>
    </person> </author>
  <cite id="c200" format="IEEE"/>
</paper>
...
```

Table 6.2: A valid XML document conforming to the DTD for `Conference` of Table 6.1.

Towards conversion between XML and relational models, an array of research has addressed the particular issues lately. On the commercial side, database vendors are busily extending their databases to adopt XML types. Typically, they can handle XML data using BLOB/CLOB formats along with a limited keyword searching or using some object-relational features [CX00, BKK00], but not many details have been revealed. On the research side, following works are related to our work:

- **Structure-oriented XML to Relational conversion**: Work done in STORED [DFS98] is one of the first significant and concrete attempts to this end and deals with non-valid XML documents. STORED uses a data mining technique to find a representative DTD whose support exceeds the pre-defined threshold and convert XML documents to relational format using the DTD. [Bou99] discusses template language-based transformation from DTD to relational schema which requires human experts to write an XML-based transformation rule. [STH99] presents three inlining algorithms that focus on the table level of the schema conversions. On the contrary, [FK99] studies different performance issues among eight algorithms that focus on the attribute and value level of the schema. [SYU99] proposes a DTD-independent mapping algorithm. While ignoring specific characteristics hidden in each DTD, [SYU99] decomposes XML documents into element, attribute, text and path tables, so that the changes of DTDs of the XML documents do not necessarily result in invalid mapping as found in examples [DFS98, STH99]. Since our CPI algorithm provides a systematic way of finding and preserving constraints from a DTD, ours is an improvement to the existing transformation algorithms. Recent work in [KKR00] attempts a conversion approach based on the notion of meta schema between XML and relational models, but mainly focuses on the structural mapping unlike ours.

More recently, [BFR02] attacks the XML to relational conversion problem from the cost-based approach, where the best conversion configuration among many candidates are chosen based on the cost estimates obtained through XML data statistics and XQuery [CFR01] workload. Unlike other algorithms, the converted relational schemas from the identical XML schema vary depending on the applications. However, this approach still focuses only on the structural conversion aspect.

- **Constraints-oriented XML to Relational conversion**: [LC00b] proposes a method where the hidden semantic constraints in DTD are systematically found and translated into relational formats. Since the method is orthogonal to the structure-oriented conversion methods, it can be used along with algorithms [DFS98, Bou99, STH99, FK99] with little change. We are not aware of any other work on this problem. This Chapter is an extended work of [LC00b].

6.3 Transforming DTD to Relational Schema

Transforming a hierarchical XML model to a flat relational model is not a trivial task. There are several difficulties including non 1-to-1 mapping, set values, recursion, and fragmentation issues [STH99]. For a better presentation, we chose one particular transformation algorithm, called the *hybrid inlining algorithm* [STH99] among many algorithms [Bou99, DFS98, FK99, STH99]. It is chosen since it exhibits the pros of the other two competing algorithms in [STH99] without severe side effects and it is a more generic algorithm than those in [Bou99, DFS98].

Since issues of discovering and preserving semantic constraints in this Chapter is independent of that of transformation algorithms, our technique can be applied to other transformation algorithms easily.

6.3.1 Choice Elimination Algorithm

Before describing the *hybrid* algorithm, let us first discuss an algorithm that eliminates the choice operators (|) from the content models of a DTD while trying to maintain the same semantics. [STH99] does not provide any details on this subtle but important issue and simply assumes that such pre-processing has been already done.

The choice operators are heavily used in XML model, but are not natively supported in relational model. For instance `<!ELEMENT r (a|b)>` in XML model implies that "`r` can have either `a` or `b` but not both at the same time". Translating this to relational model, the closest mapping with the same semantics would be having a table "r" with two nullable columns "a" and "b", (i.e., `(a?,b?)`) with a constraint enforcing one of the two columns must be null at all times as follows:

```
CREATE TABLE r (
 a     VARCHAR(20),
 b     VARCHAR(20),
 CHECK ((a is NOT NULL AND b is NULL) OR (a is NULL AND b is NOT NULL))
);
```

Hence, when there is no nested content models, any arbitrary long content models with | operators `<!ELEMENT r (`a_1 `| … |` a_n`)>` can be treated as if it were `<!ELEMENT r (`a_1`?, …,` a_n`?)>` with an additional constraint like `CHECK ((`a_1 `is NOT NULL AND` a_2 `is NULL AND …` a_n `is NULL) OR … OR (`a_1 `is NULL AND` a_2 `is NULL AND …` a_n `is NOT NULL))`. Let us call this mapping heuristics

```
Input   : Regular expression $r$
Output: Regular expression $(r_1|r_2|\ldots|r_n)$ equivalent to $r$
switch $r$ do
    case $r$ does not contain "|" operator
        return $r$;
    case $r = (r_1)^*$
        migrateChoice($r_1$) $= (a_1|a_2|\ldots|a_n)$;
        return $(a_1^*, a_2^*, \ldots, a_n^*)^*$;
    case $r = (r_1|r_2)$
        migrateChoice($r_1$) $= (a_1|a_2|\ldots|a_n)$;
        migrateChoice($r_2$) $= (b_1|b_2|\ldots|b_n)$;
        return $(a_1|a_2|\ldots|a_n|b_1|b_2|\ldots|b_n)$;
    case $r = (r_1, r_2)$
        migrateChoice($r_1$) $= (a_1|a_2|\ldots|a_n)$;
        migrateChoice($r_2$) $= (b_1|b_2|\ldots|b_n)$;
        return $((a_1,b_1)|(a_1,b_2)|\ldots|(a_1,b_n)|(a_2,b_1)|(a_2,b_2)|\ldots|(a_2,b_n)|$
        $\ldots|(a_n,b_1)|(a_n,b_2)|(a_n,b_n))$;
```

Algorithm 1: `migrateChoice`

as `convertChoice()`.

Now consider a general case where a content model can in turn contain further nested content models in it and all use | operators in a complex manner. From a basic regular expression algebraic law ([HMU01], page 118), the following equality holds: `(a | b)* = (a*, b*)*`. Using the law, the shown Algorithm `migrateChoice()` determines an equivalent regular expression of the form $(r_1|r_2|\ldots|r_n)$, where no r_i $(1 \leq i \leq n)$ contains | operator (i.e., remove | in inner groups except ones in the outermost group)

Once we have a content model returned from `migrateChoice()`, then all | operators have migrated from inside to outside. Next step is to flatten content models out. For instance, [STH99] describes various heuristics such as `a*? = a*` or `(a*,a*) = (a*)`. Let us call such steps as `flatten()`.

As a conclusion, content models of DTDs using the choice operator can be in general converted to relational schema by going through 1) `migrateChoice(`r`)` for each content model r and 2) successively `convertChoice(`r`)` and 3) `flatten(`r`)`. For further details of the algorithm, refer to [MLM01b].

Example 7. Consider `<!ELEMENT r ((a|b)*|c)>`. First, `migrateChoice(`r`)` is converted to `migrateChoice(`$(a|b)^*|c)$`)` and is in turn converted to two calls: `migrateChoice(`$(a|b)^*$`)` and `migrateChoice(`c`)`. Further, `migrateChoice(`$(a|b)^*$`)` returns `migrateChoice(`$(a^*, b^*)^*$`)` while `migrateChoice(`c`)` remains intact. Hence, eventually `migrateChoice(`r`)` returns `migrateChoice(`$(a^*, b^*)^*|c$`)` At second stage, $(a^*, b^*)^*|c$ is simulated by $((a^*, b^*)^*?, c?)$ by `convertChoice()` and in turn simplified to $((a^*, b^*)^*, c?)$ by `flatten()`, generating `<!ELEMENT r (a*,b*,c?)>` with a proper constraint at the end. The new content model is free of the choice operator and can be fed into the hybrid algorithm in the next section. □

6.3.2 Hybrid Inlining Algorithm

The *hybrid* algorithm [STH99] essentially does the following[1]:

1. Create a *DTD graph* that represents the structure of a given DTD. A DTD graph can be constructed when parsing the given DTD. Its nodes are elements, attributes, or operators in the DTD. Each element appears exactly once in the graph, while attributes and operators appear as many times as they appear in the DTD.

2. Identify *top nodes* in a DTD graph. A top node satisfies any of the following conditions: 1) not reachable from any nodes (e.g., source node), 2) direct child of "`*`" or "`+`" operator node, 3) recursive node with indegree > 1, or 4) one node between two mutually recursive nodes with indegree $= 1$. Then, starting from a top node T, *inline* all the elements and attributes at *leaf nodes* reachable from T unless they are other top nodes.

3. Attribute names are composed from the concatenated path from the top node to the leaf node using "_" as a delimiter. Use an attribute with `ID` type as a key if provided. Otherwise, add a system-generated integer key[2].

4. If a table corresponds to the shared element with indegree > 1 in the DTD, then add a field `parent_elm` to denote the parent element to which the current tuple belongs. Further, for each shared element, a new field `fk_`X is added as a *foreign key* to record the key values of parent element X. If X is inlined into another element Y, then record the Y's key value in the `fk_`Y field.

[1]We have made a few changes to the hybrid algorithm for a better presentation (e.g., renaming, supporting "|" operator), but the crux of the algorithm remains intact.

[2]In practice, even if there is an attribute with `ID` type, one may decide to have a system-generated key for better performance.

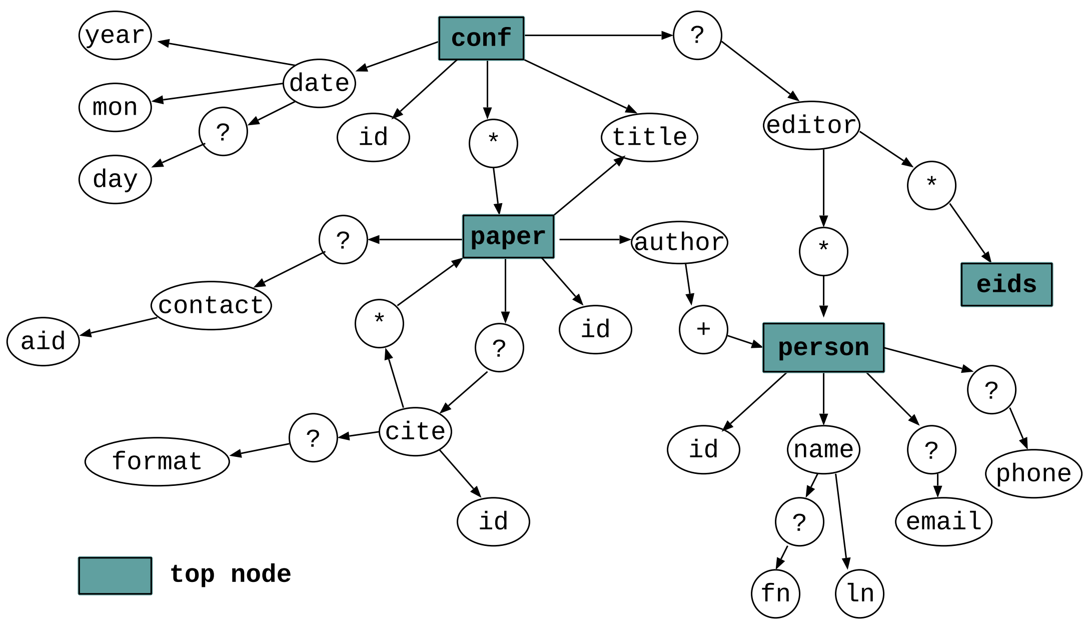


Figure 6.2: A DTD graph for the DTD in Table 6.1.

5. Inlining an element Y into a table r, corresponding to another element X (i.e., top node), creates a problem when an XML document is rooted at the element Y. To facilitate queries on such elements, a new field `root_elm` is added to a table r.

6. If an *ordered* DTD model is used, a field `ordinal` is added to record position information of sub-elements in the element. (For simplification, the `ordinal` field is not shown in this Chapter.)

For further details of the algorithm, refer to [STH99]. Figure 6.2 illustrates a DTD graph that is created from the DTD of Table 6.1. Table 6.3 shows the output of the transformation by the hybrid algorithm. Note that the hybrid algorithm does not generate *semantic constraints* (Δ).

Among eleven elements in the DTD of Table 6.1, four elements – `conf`, `paper`, `person`, and `eids` – are top nodes and thus, chosen to be mapped to the different

conf

id	root_elm	title	date_year	date_mon	date_day
er05	conf	ER	2005	May	20

conf_editor_eids

id	root_elm	fk_conf	eids
100001	conf	er05	sheth
100002	conf	er05	bossy

paper

id	root_elm	parent_elm	fk_conf	fk_cite	title	contact_aid	cite_id	cite_format
p1	conf	conf	er05	–	Indexing ...	dao	–	–
p2	conf	conf	er05	–	Logical ...	shah	c100	ACM
p3	conf	cite	–	c100	Making ...	–	–	–
p7	paper	–	–	–	Constraints ...	lee	c200	IEEE

person

id	root_elm	parent_elm	fk_conf	fk_paper	name_fn	name_ln	email	phone
klavans	conf	editor	er05	–	Judith	Klavans	klavans...	–
dao	conf	paper	–	p1	Tuong	Dao	–	–
shah	conf	paper	–	p2	Kshitij	Shah	–	–
sheth	conf	paper	–	p2	Amit	Sheth	amit@cs...	–
bossy	conf	paper	–	p3	Marcia	Bossy	–	391.4337
lee	paper	paper	–	p7	Dongwon	Lee	dongwon...	–

Table 6.3: A relational scheme ($\mathbb{S}$) along with the associated data that are converted from the DTD of Table 6.1 and XML document of Table 6.2 by the hybrid algorithm.

tables. For the top node conf, the elements date, title, and editor are reachable and thus inlined. Then, the id attribute is used as a key and the root_elm field is added. For the top node paper, the elements title, contact_aid, author, cite_format and cite_id are reachable and inlined. Since the paper element is shared by the conf and cite elements (two incoming edges in a DTD graph), new fields parent_elm, fk_conf and fk_cite are added to record who and where the parent node was. Note that in the paper table (Table 6.3), a tuple with id="p7" has the value "paper" for the root_elm field. This is because the element <paper id="p7"> is rooted in the DTD (Table 6.2) without being embedded in other elements. Consequently, its parent_elm, fk_conf and fk_cite fields are null. For the top node person, the elements name_fn, name_ln and email are reachable and inlined. Since the person is shared by the author and editor elements, again, the parent_elm is added. Note that in the person table (Table 6.3), a tuple with id="klavans" has the value "editor", not "paper", for the parent_elm field. This implies that "klavans" is in fact an editor, not an author of the paper.

6.4 Semantic Constraints in DTDs

6.4.1 Domain Constraints

When the domain of the attributes is restricted to a certain specified set of values, it is called *Domain Constraints*. For instance, in the following DTD, the domain of the attributes gender and married are restricted.

```
<!ATTLIST author gender  (male|female) #REQUIRED
                 married (yes|no)      #IMPLIED>
```

In transforming such DTD into relational schema, we can enforce the domain constraints using SQL CHECK clause as follows:

```
CREATE DOMAIN gender VARCHAR(10) CHECK (VALUE IN ("male", "female"))
CREATE DOMAIN married VARCHAR(10) CHECK (VALUE IN ("yes", "no"))
```

When the mandatory attribute is defined by the `#REQUIRED` keyword in the DTD, it needs to be forced in the transformed relational schema as well. That is, the attribute `ln` cannot be omitted below.

```
<!ELEMENT person  EMPTY>
<!ATTLIST person  fn CDATA  #IMPLIED   ln CDATA  #REQUIRED>
```

We use the notation "$X \nrightarrow \emptyset$" to denote that an attribute X cannot be null. This kind of domain constraint can be best expressed by using the `NOT NULL` clause in SQL as follows:

```
CREATE TABLE person (fn VARCHAR(20), ln VARCHAR(20) NOT NULL)
```

6.4.2 Cardinality Constraints

In a DTD declaration, there are only 4 possible cardinality relationships between an element and its sub-elements as illustrated below:

```
<!ELEMENT article (title, author+, reference*, price?)>
```

(0,1). ("at most" semantics): An element can have either zero or one sub-element. (e.g., sub-element `price`)

(1,1). ("only" semantics): An element must have one and only one sub-element. (e.g., sub-element `title`)

(0,N). ("any" semantics): An element can have zero or more sub-elements. (e.g., sub-element `reference`)

(1,N). ("at least" semantics): An element can have one or more sub-elements. (e.g., sub-element `author`)

Following the notations in [BCN92], let us call each cardinality relationship as type (0,1), (1,1), (0,N), (1,N), respectively. From these cardinality relationships, mainly three constraints can be inferred. First, whether or not the sub-element can be null. Similar to the attribute case, we use the notation "$X \nrightarrow \emptyset$" to denote that an element X cannot be null. This constraint is easily enforced by the `NULL` or `NOT NULL` clause. Second, whether or not more than one sub-element can occur. This is also known as *singleton constraint* in [Woo99] and is one kind of equality-generating dependencies. Third, given an element, whether or not its sub-element should occur. This is one kind of tuple-generating dependencies. The second and third types will be further discussed below.

6.4.3 Inclusion Dependencies (INDs)

An *Inclusion Dependency* assures that values in the columns of one fragment must also appear as values in the columns of other fragments and is a generalization of the notion of *referential integrity*.

Trivial form of INDs found in the DTD is that "given an element X and its sub-element Y, Y must be included in X (i.e., $Y \subseteq X$)". For instance, from the `conf` element and its four sub-elements in DTD, the following INDs can be found as long as `conf` is not null: {`conf.title` $\subseteq$ `conf`, `conf.date` $\subseteq$ `conf`, `conf.editor` $\subseteq$ `conf`, `conf.paper` $\subseteq$ `conf`}. Another form of INDs can be found in the attribute definition part of the DTD with the use of the `IDREF(S)` keyword. For instance, consider the `contact` and `editor` elements in the DTD in Table 6.1 shown below:

```
<!ELEMENT person  (name,(email|phone)?>
<!ATTLIST person  id   ID     #REQUIRED>
<!ELEMENT contact EMPTY>
<!ATTLIST contact aid  IDREF  #REQUIRED>
<!ELEMENT editor  (person*)>
<!ATTLIST editor  eids IDREFS #IMPLIED>
```

The DTD restricts the `aid` attribute of the `contact` element such that it can only point to the `id` attribute of the `person` element[3]. Further, the `eids` attribute can only point to multiple `id` attributes of the `person` element. As a result, the following INDs can be derived: $\{$`editor.eids` $\subseteq$ `person.id, contact.aid` $\subseteq$ `person.id` $\}$. INDs can be best enforced by the "foreign key" concept if the attribute being referenced is a primary key. Otherwise, it needs to use the `CHECK`, `ASSERTION`, or `TRIGGERS` facility of SQL.

6.4.4 Equality-Generating Dependencies (EGDs)

The *Singleton Constraint* [Woo99] restricts an element to have "at most" one sub-element. When an element type X satisfies the singleton constraint towards its sub-element type Y, if an element instance x of type X has *two* sub-elements instances y_1 and y_2 of type Y, then y_1 and y_2 must be the same. This property is known as *Equality-Generating Dependencies (EGDs)* and denoted by "$X \rightarrow Y$" in database theory. For instance, two EGDs: $\{$`conf` $\rightarrow$ `conf.title, conf` $\rightarrow$ `conf.date`$\}$ can be derived from the `conf` element of Table 6.1. This kind of EGDs can be enforced by SQL `UNIQUE` construct. In general, EGDs occur in the case of the (0,1) and (1,1) mappings in the cardinality constraints.

[3]Precisely, an attribute with `IDREF` type does not specify which element it should point to. This information is available only by human experts. However, new XML schema languages such as XML-Schama and DSD can express where the reference actually points to [LC00a].

6.4.5 Tuple-Generating Dependencies (TGDs)

Tuple-Generating Dependencies (TGDs) in a relational model require that some tuples of a certain form be present in the table and use the "$\twoheadrightarrow$" symbol. Two useful forms of TGDs from DTD are the *child* and *parent constraints* [Woo99].

1. **Child constraint:** `"Parent` $\twoheadrightarrow$ `Child"` states that every element of type *Parent* must have at least one child element of type *Child*. This is the case of the (1,1) and (1,N) mappings in the cardinality constraints. For instance, from the DTD in Table 6.1, because the `conf` element must contain the `title` and `date` sub-elements, the child constraint `conf` $\twoheadrightarrow$ {`title, date`} holds.

2. **Parent constraint:** `"Child` $\twoheadrightarrow$ `Parent"` states that every element of type *Child* must have a parent element of type *Parent*. According to XML specification, XML documents can start from any level of elements without necessarily specifying its parent element, when a root element is not specified by `<!DOCTYPE root>`. In the DTD of Table 6.1, for instance, the `editor` and `date` elements can have the `conf` element as their parent. Further, if we know that all XML documents were started at the `conf` element level, rather than the `editor` or `date` level, then the parent constraint {`editor, date`} $\twoheadrightarrow$ `conf` holds. Note that the `title` $\twoheadrightarrow$ `conf` does not hold since the `title` element can be a sub-element of either the `conf` or `paper` element.

6.5 Constraints-Preserving Inlining Algorithm

To help find semantic constraints, we use the following data structure:

Definition 11 *An* ***annotated DTD graph (ADG)*** $\mathbb{G}$ *is a pair* $(\mathbb{V}, \mathbb{E})$*, where*

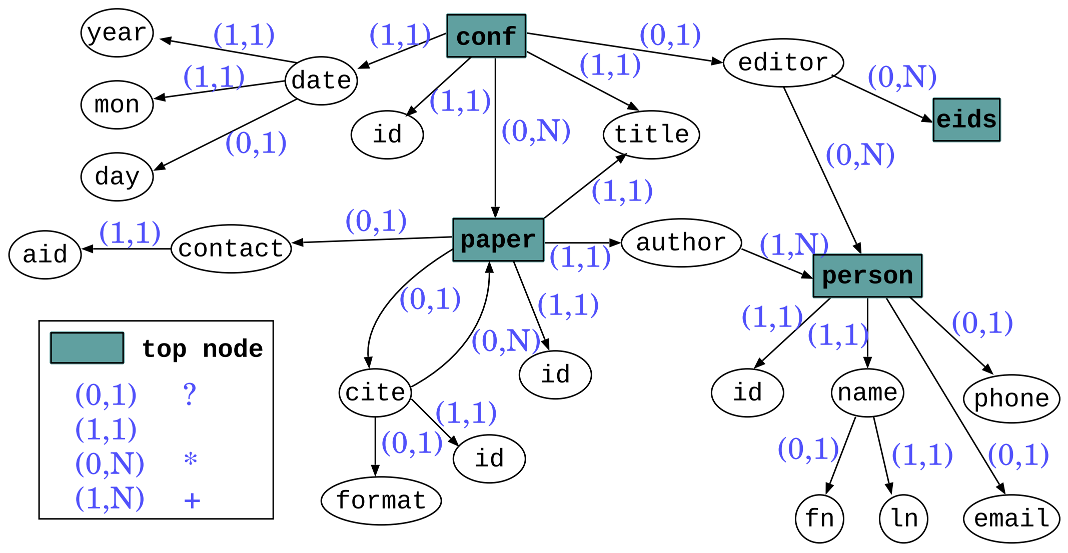


Figure 6.3: An *Annotated DTD graph* for the `Conference` DTD of Table 6.1.

$\mathbb{V}$ is a finite set and $\mathbb{E}$ is a binary relation on $\mathbb{V}$. The set $\mathbb{V}$ consists of element and attributes in a DTD. Each edge $e \in \mathbb{E}$ is labeled with the cardinality relationship types as defined in Section 6.4.2. In addition, each vertex $v \in \mathbb{V}$ carries the following information:

1. ***`indegree`*** *stores the number of incoming edges.*

2. ***`type`*** *contains the element type name in the content model of the DTD (e.g., **`conf`** or **`paper`**).*

3. ***`tag`*** *stores a flag value whether the node is an element or attribute (if attribute, it contains the attribute keyword like `ID` or `IDREF`, etc.).*

4. ***`status`*** *contains "visited" flag if the node was visited in a depth-first search or "not-visited".* □

Note that the cardinality relationship types in ADG considers not only element vs. sub-element relationships but also element vs. attribute relationships. For instance, from the DTD <`!ATTLIST X Y #IMPLIED Z #REQUIRED`>,

Relationship	Symbol	Semantics	not null	EGDs	TGDs
(0,1)	?	at most	no	yes	no
(1,1)		only	yes	yes	yes
(0,N)	*	any	no	no	no
(1,N)	+	at least	yes	no	yes

Table 6.4: Cardinality relationships and their corresponding semantic constraints.

two types of cardinality relationships (i.e., type (0,1) between element X and attribute Y, and type (1,1) between element X and attribute Z) can be derived. Figure 6.3 illustrates an example of ADG for the `Conference` DTD of Table 6.1. Then, the cardinality relationships can be used to find semantic constraints in a *systematic* fashion. Table 6.4 summarizes 3 main semantic constraints that can be derived from and the `findConstraints()` algorithm below is immediately derived from Table 6.4.

```
Input   : Node $v$ and $w$
switch edge($v$,$w$) do
    case type (0,1)
        $v \rightarrow w$;
    case type (1,1)
        $w$ is not null; $v \rightarrow w$; $v \twoheadrightarrow w$;
    case type (0,N)
        /* empty */;
    case type (1,N)
        $w$ is not null; $v \twoheadrightarrow w$;
```

Algorithm 2: `findConstraints`

Semantic constraints discovered by `findConstraints()` have additional usage as discussed in Section [LC01]. However, to enforce correct semantics in the newly generated relational schema, the semantic constraints in XML terms need to be rewritten in relational terms. This is done by the algorithm `rewriteConstraints()`.

We shall now describe our complete DTD-to-relational schema transformation algorithm: *CPI (Constraints-preserving Inlining) algorithm* is a combination of the hybrid inlining, `findConstraints()` and `rewriteConstraints()` algorithms. The CPI algorithm is illustrated in `CPI()` and `hybrid()`.

The algorithm first identifies all the top nodes from the ADG. This can be done using algorithms to find sources or strongly-connected components in a graph [STH99]. Then, for each top node, the algorithm generates a corresponding table scheme using `hybrid()`. The associated constraints are found and rewritten in relational terms using `findConstraints()` and `rewriteConstraints()`, respectively. The `hybrid()` algorithm scans an ADG in a depth-first search while finding constraints. The final output schema is the union of all the table schemes and semantic constraints.

Table 6.5 contains the semantic constraints that are rewritten from XML terms to relational terms. As an example, the CPI algorithm will eventually spit out the following SQL `CREATE` statement for the `paper` table. Note that not only is the relational scheme provided, but the semantic constraints are also ensured by use of the `NOT NULL`, `KEY`, `UNIQUE` or `CHECK` constructs.

```
CREATE TABLE paper (
  id          NUMBER      NOT NULL,
  title       VARCHAR(50) NOT NULL,
  contact_aid VARCHAR(20),
  cite_id     VARCHAR(20),
Input  : Constraints $\Delta'$ in XML notation
Output: Constraints $\Delta$ in relational notation
switch $\Delta'$ do
    case $X \not\twoheadrightarrow \emptyset$
        If $X$ is mapped to attribute $X'$ in table scheme $A$, then $A[X']$
        cannot be null. (i.e., "CREATE TABLE $A$ (...$X'$ NOT NULL...)") ;
    case $X \subseteq Y$
        If $X$ and $Y$ are mapped to attributes $X'$ and $Y'$ in table scheme $A$
        and $B$, respectively, then rewrite it as $A[X'] \subseteq B[Y']$. (i.e., If $Y'$ is
        a primary key of $B$, then "CREATE TABLE $A$ (...FOREIGN KEY ($X'$)
        REFERENCES $B(Y')$...)". Else "CREATE TABLE $A$ (...($X'$) CHECK ($X'$
        IN (SELECT $Y'$ FROM $B$))...") ;
    case $X \rightarrow X.Y$
        If element $X$ and $Y$ are mapped to the same table scheme $A$ (i.e.,
        since $Y$ is not a top node, $Y$ becomes an attribute of table $A$) and
        $Z$ is the key attribute of $A$, then rewrite it as $A[Z] \rightarrow A[Y]$. (i.e.,
        "CREATE TABLE $A$ (...UNIQUE ($Y$), PRIMARY KEY ($Z$)...)");
    case $X \twoheadrightarrow X.Y$
        if (element $X$ and $Y$ are mapped to the same table) then
            Let $A$ be the table and $Z$ be the key attribute of $A$. Then
            rewrite it as $A[Z] \twoheadrightarrow A[Y]$. (i.e., "CREATE TABLE $A$ (...$Y$ NOT
            NULL, PRIMARY KEY ($Z$)...)") ;
        else
            Let the tables be $A$ and $B$, respectively and $Z$ be the key at-
            tribute of $A$. Then rewrite it as $B[fk\_A] \subseteq A[Z]$. (i.e., "CREATE
            TABLE $B$ (...FOREIGN KEY ($fk\_A$) REFERENCES $A(Z)$...)")
return $\Delta$;
```

Algorithm 3: `rewriteConstraints`

```
Input   : Annotated DTD Graph $\mathbb{G} = (V, E)$
Output: Relational Schema $\mathbb{R}$
$\mathbb{V} \leftarrow$ topnode($\mathbb{G}$);
for each $v \in \mathbb{V}$ do
    $table\_def \leftarrow \{\}$;
    if $v.tag$ = 'element' then
        add('root_elm', $table\_def$); /* start where?  */
    if $v.indegree > 1$ then
        add('parent_elm', $table\_def$);   /* shared elements case */
        add(concat('fk_', parent($v$)), $table\_def$);
    $\mathbb{W} \leftarrow Adj[v]$; w $\in \mathbb{W}$;
    if any $w.tag$ = 'ID' then add($w.type$, $table\_def$);
    else add('id', $table\_def$);   /* system-generated primary key */
    $\mathbb{R} \leftarrow \mathbb{R}$ + hybrid($v$, $table\_def$, $\emptyset$);
return $\mathbb{R}$;
```

Algorithm 4: `CPI`

```
Input   : Vertex $v$, TableDef $table\_def$, string $attr\_name$
Output: Relational Schema $\mathbb{R}$
$v.status \leftarrow$ 'visited';
for each $w \in Adj[v]$ do
    if $w.status$ = 'not-visited' then
        $\Delta' \leftarrow$ findConstraints($v$, $w$); $\Delta \leftarrow$ rewriteConstraints($\Delta'$);
        hybrid($w$, $table\_def$, concat($attr\_name$, '_', $w.type$));
add($attr\_name$, $table\_def$); $\mathbb{R} \leftarrow table\_def + \Delta$;
return $\mathbb{R}$;
```

Algorithm 5: `hybrid`

Type	Semantic constraints in relational notation
ID	conf_editor_eids[eids] $\subseteq$ person[id], paper[contact_aid] $\subseteq$ person[id]
EGD	conf[id] $\rightarrow$ conf[title,date_year,date_mon,date_day] paper[id] $\rightarrow$ conf[title,contact_aid,cite_id,cite_format] person[id] $\rightarrow$ conf[name_fn,name_ln,email]
TGD	conf[id] $\twoheadrightarrow$ conf[title,date_year,date_mon,date_day] paper[id] $\twoheadrightarrow$ conf[title,contact_aid,cite_id,cite_format] person[id] $\twoheadrightarrow$ conf[name_fn,name_ln,email] conf_editor_eids[fk_conf] $\subseteq$ conf[id] paper[fk_conf] $\subseteq$ conf[id], paper[fk_cite] $\subseteq$ paper[cite_id] person[fk_conf] $\subseteq$ conf[id], person[fk_paper] $\subseteq$ paper[id]
not null	conf[id,title,date_year,date_mon,root_elm] $\not\twoheadrightarrow \emptyset$ conf_editor_eids[id,root_elm] $\not\twoheadrightarrow \emptyset$ paper[id,title,root_elm] $\not\twoheadrightarrow \emptyset$, person[id,name_ln,root_elm] $\not\twoheadrightarrow \emptyset$

Table 6.5: The semantic constraints in relational notation for the `Conference` DTD of Table 6.1.

```
  cite_format VARCHAR(50) CHECK (VALUE IN ("ACM", "IEEE")),
  root_elm    VARCHAR(20) NOT NULL,
  parent_elm  VARCHAR(20),
  fk_cite     VARCHAR(20)
      CHECK (fk_cite IN (SELECT cite_id FROM paper)),
  fk_conf     VARCHAR(20),
  PRIMARY KEY (id),
  UNIQUE (cite_id),
  FOREIGN KEY (fk_conf)      REFERENCES conf(id),
  FOREIGN KEY (contact_aid)  REFERENCES person(id)
);
```

6.6 Experimental Results

We have implemented the CPI algorithm in Java using the IBM XML4J package. Table 6.6 shows a summary of our experimentation. We gathered test DTDs from "http://www.oasis-open.org/cover/xml.html" and [Sah00]. Since some DTDs had syntactic errors caught by the XML4J, we had to modify them manually. Note that people seldom used the `ID` and `IDREF(S)` constructs in their DTDs except the `XMI` and `BSML` cases. The number of tables generated in the relational schema was usually smaller than that of elements/attributes in DTDs due to the inlining effect. The only exception to this phenomenon was the `XMI` case, where extensive use of types (0,N) and (1,N) cardinality relationships resulted in many top nodes in the ADG.

The number of semantic constraints had a close relationship with the design of the DTD hierarchy and the type of cardinality relationship used in the DTD. For instance, the `XMI` DTD had many type (0,N) cardinality relationships, which do not contribute to the semantic constraints. As a result, the number of semantic constraints at the end was small, compared to that of elements/attributes in the DTD. This was also true for the `OSD` case. On the other hand, in the `ICE` case, since it used many type (1,1) cardinality relationships, it resulted in many semantic constraints.

6.7 Summary

This Chapter presents a method to transform XML DTD to relational schema both in *structural* and *semantic* aspects. After discussing the semantic constraints hidden in DTDs, two algorithms are presented for: 1) discovering the semantic constraints using the hybrid inlining algorithm, and 2) rewriting the

DTD Semantics		DTD Schema		Relational Schema			
Name	Domain	Elm/Attr	ID/IDREF(S)	Table/Attr	$\rightarrow$	$\twoheadrightarrow$	$\not\twoheadrightarrow \emptyset$
`novel`	literature	10/1	1/0	5/13	6	9	9
`play`	Shakespeare	21/0	0/0	14/46	17	30	30
`tstmt`	religious text	28/0	0/0	17/52	17	22	22
`vCard`	business card	23/1	0/0	8/19	18	13	13
`ICE`	content synd.	47/157	0/0	27/283	43	60	60
`MusicML`	music desc.	12/17	0/0	8/34	9	12	12
`OSD`	s/w desc.	16/15	0/0	15/37	2	2	2
`PML`	web portal	46/293	0/0	41/355	29	36	36
`Xbel`	bookmark	9/13	3/1	9/36	9	1	1
`XMI`	metadata	94/633	31/102	129/3013	10	7	7
`BSML`	DNA seq.	112/2495	84/97	104/2685	99	33	33

Table 6.6: Experimental results of CPI algorithm.

semantic constraints in relational notation. Our experimental results reveal that constraints can be systematically preserved during the conversion from XML to relational schema. Such constraints can also be used for semantic query optimization or semantic caching.

Despite the obstacles in converting from XML to relational models and vice versa, there are several practical benefits:

- Considering the present market that is mostly dominated by RDB products, it is not easy nor practical to abandon RDB to support XML. It is very likely that industries would be reluctant to adopt the new technology if it does not support the existing RDB techniques as they were reluctant towards object-oriented database in the past.
- By using RDB as an underlying storage system, the mature RDB techniques can be leveraged. That is, a vast number of sophisticated techniques (e.g.,

OLAP, Data Mining, Data Warehousing, etc.) developed for RDB can be applied to XML data with minimal changes.

- The integration of a large amount of XML data on the Web with the legacy data in relational format is possible.

We strongly believe that devising more accurate and efficient conversion methodologies between XML and relational models is very important and our CPI algorithm can serve as an enhancement for such conversion algorithms.

Due to many benefits from using relational databases as storage systems for XML data, the need for efficient and effective conversion between relational and XML models will significantly grow in a foreseeable future. We believe the following directions of research are very important.

First, as we move to more expressive next generation XML schema languages such as XML-Schema [Fal01] or RELAX [Mur00b], the degree of complexities captured in a XML schema is far greater than that in a DTD. For instance, XML-Schema supports an extensive set of features to specify structural and semantic constraints. However, all existing XML to relational conversion algorithms (discussed in Section 6.2) focus only on the DTD case, which is the simpliest and least expressive XML schema language according to [LC00a, LMM00]. Therefore, there is an immediate need to modify and extend the current conversion algorithms to support more complex schema languages.

Second, with XML emerging as the data format of the Internet era, there is a substantial increase in the amount of data encoded in XML. However, the majority of everyday data is still stored and maintained in relational databases. Therefore, we expect the needs to convert such relational data into XML documents to grow substantially as well. Although commercial database vendors already support tools that generate XML documents out of relational data, the types of XML documents generated are very simple in their structure and consequently cannot capture all semantics in the original relational schema. For instance, a majority of tools can only convert the so-called “flat translation” where a table t and columns c_i of relational model is mapped to an element e and its attributes a_i of XML model. We have proposed the “nesting-based translation” to capture certain semantics in the original relational schema [LMC01], however, more research efforts in that direction are needed.

CHAPTER 7

Relational to XML Conversion

Two algorithms, called NeT and CoT, to translate relational schemas to XML schemas using various semantic constraints are presented. The XML schema representation we use is a language-independent formalism named *XSchema*, that is both precise and concise. A given *XSchema* can be mapped to a schema in any of the existing XML schema language proposals. Our proposed algorithms have the following characteristics: (1) NeT derives a nested structure from a flat relational model by repeatedly applying the *nest* operator on each table so that the resulting XML schema becomes hierarchical, and (2) CoT considers not only the structure of relational schemas, but also semantic constraints such as inclusion dependencies during the translation - it takes as input a relational schema where multiple tables are interconnected through inclusion dependencies and converts it into a *good XSchema*. To validate our proposals, we present experimental results using both real schemas from the UCI repository and synthetic schemas from TPC-H.

7.1 Background

XML [BPS00] is rapidly becoming one of the most widely adopted technologies for information exchange and representation on the World Wide Web. With XML emerging as *the* data format of the Internet era, there is a substantial increase in the amount of data encoded in XML. However, the majority of everyday data is still stored and maintained in relational databases. Therefore, we expect the needs to convert such relational data into XML documents will grow substantially as well. In this Chapter, we study the problems in this conversion. Especially, we are interested in finding XML schema[1] (e.g., DTD [BPS00], RELAX-NG [CM01], XML-Schema [TBM01]) that *best* describes the existing relational schema. Having an XML schema that precisely describes the semantics and structures of the original relational data is important to further maintain the converted XML documents in future.

At present, there exist several tools that enable the composition of XML documents from relational data, such as XML Extender from IBM[2], XML-DBMS[3], DB2XML [Tur99], SilkRoute [FTS00], and XPERANTO [CFI00]. In these tools, the success of the conversion is closely related with the quality of the target XML schema onto which a given input relational schema is mapped. However, the mapping from the relational schema to the XML schema is specified by human experts. Therefore, when large amount of relational schemas and data need to be translated into XML documents, a significant investment of human effort is required to initially design target schemas. To make matters worse, in the context of merging legacy relational data to existing XML documents, devising a good XML schema that does not violate existing structures and constraints is a non-trivial task. Being able to *automatically* infer a precise XML schema out of relational schema would be very useful in such settings.

In this Chapter, therefore, we are interested in finding a method that can

[1]We differentiate two terms – XML schema(s) and XML-Schema. The former refers to a general term for schema in XML model while the latter refers to one particular kind of XML schema language proposed by W3C [TBM01].

[2]http://www-4.ibm.com/software/data/db2/extenders/xmlext/

[3]http://www.rpbourret.com/xmldbms/index.htm

infer the *best* XML schema from the given relational schema automatically. We particularly focus on two aspects of the translation: (1) **Structural aspect**: We want to find the most intuitive and precise XML schema structure from the given relational schema. We especially try to use the hidden characteristics of data using *nest* operator, and (2) **Semantic aspect**: During the translation, we want to use semantic constraints that could be either acquired from database directly or provided by human experts explicitly. Let us first consider a motivating example illustrating why correctly converting structures and semantics of relational schema into XML schema is an important and useful problem.

Example 8. Suppose one tries to model an entity `paper` in a database. It would typically contain attributes `Title`, `Year`, `Author`, `Keyword`, etc. Besides, suppose there can be one or many authors and keywords per paper. In representing such model into relational schema, one in general cannot express the fact that there can be many authors and keywords in a single table since it would violate 1NF. Hence, typical relational schema that captures the scenario would be as follows:

```
paper (Pid NUMBER, Title VARCHAR, Year NUMBER,
       Author VARCHAR, Keyword VARCHAR)
```

Suppose this table is translated to an XML schema (with a common attribute `<!ATTLIST paper Pid ID #REQUIRED>`):

```
(1) <!ELEMENT paper (Title,Year,Author,Keyword)>
(2) <!ELEMENT paper (Title,Year,Author+,Keyword+)>
```

From the users' perspective, the second DTD is better than the first DTD, since it represents the given relational schema more accurately: the first DTD incorrectly implies that `paper` can have only one `Author` and `Keyword`. The main reason of this discrepancy between the users' perception and the representation is that in relational model, concepts need to be *flattened out* to fit into the model. Since XML model allows hierarchical nesting, it would be desirable to *unflatten* concepts to make such structures if preferred so. □

In short, we aim to solve the following problem:

> *Given a relational schema* $\mathbb{R}$*, find an XML schema* $\mathbb{X}$ *that best satisfies semantic constraints of* $\mathbb{R}$

We first present a straightforward relational to XML translation algorithm, called *Flat Translation* (FT). Since FT maps the flat relational model to the flat XML model in a one-to-one manner, it does not utilize the regular expression operators (e.g., "*", "+") supported in the content models of XML. Then, we present our first proposal called *Nesting-based Translation* (NeT), to remedy the problems found in FT. NeT derives nested structures from a flat relational model by the use of the *nest* operator so that the resulting XML schema is more intuitive and precise than otherwise. Although NeT infers hidden characteristics of data by nesting, it is only applicable to a single table at a time. Therefore, it is unable to capture a correct "big picture" of relational schema where many tables are interconnected. To remedy this problem, we present the second proposal called *Constraints-based Translation* (CoT); CoT considers inclusion dependencies during the translation. Such constraints can be acquired from database through ODBC/JDBC interface or provided by human experts who are familiar with the semantics of the relational schema being translated. CoT is capable of generating a more intuitive XML schema than what NeT. Figure 7.1 illustrates the overview of our approach.

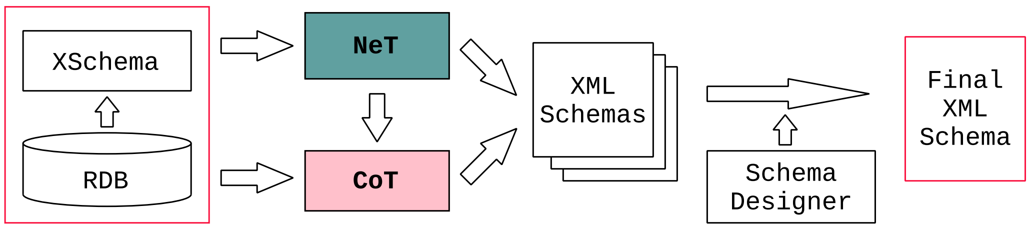


Figure 7.1: Overview of NeT and CoT algorithms.

7.2 Related Work

There also have been work in mapping from non-relational models to XML model. [MLM01a] studies the conversion from XML to ER model and vice versa. Generation of an XML schema from a UML model is studied in [NEF00]. In addition, there have been other DTD inference algorithms that take as "input" a set of XML documents [GGR00] or a view description [PV99].

Conversion from relational to XML models can ba categorized as follows:

- **Structure-oriented Relational to XML conversion**: Some primitive work has been done in [Tur99] dealing with the transformation from relational tables to XML documents. In XML Extender, the user specifies the mapping through a language such as DAD or XML Extender Transform Language. In XML-DBMS, a template-driven mapping language is provided to specify the mappings. SilkRoute [FTS00] provides a declarative query language (RXL) for viewing relational data in XML. Applications express the answer data as a query over the view and SilkRoute dynamically materializes the fragment of an XML view. [SSB00] extends SQL to specify the conversion process declaratively, whereas SilkRoute proposes a new language RXL and describes an extensive study on the issues of efficiently implementing the algorithms. Similar to SilkRoute, XPERANTO [CFI00] aims to provide a uniform XML interface to underlying ORDB, transparently providing an XML-to-SQL query rewriter and a table-to-XML answer converter. Its output XML view is, however, mainly specified by the user's input XML queries. Note that in SilkRoute and XPERANTO, the user has to specify the query in the appropriate query language. DB2XML uses an algorithm similar to FT (and hence suffers from similar problems).

- **Constraints-oriented Relational to XML conversion**: *Path constraints* on a semi-structured model [BFW98] or XML model [FS00] have been studied, mostly with respect to their implication problems. However, to our best knowledge, there has not been much work on this direction of conversion problem. For instance, [FS00] proposes three languages to capture the semantics of XML model and presents implication results, but does not deal with issues on converting constraints from RDB to XML model. Recently, the authors proposed to convert relational schema to XML schema using the hidden data semantics found by the *nest* operator in [LMC01].

7.3 Input and Output Models

We first briefly define the input and output models for the translation. In relational databases, schema is typically created by SQL DDL (e.g., `CREATE`) statements. Therefore, by examining such DDL statements, one can find out the original schema information. Even if such DDL statements are not available, one can still infer the schema information - table and column names, key and foreign key information, etc - by querying the database through an ODBC/JDBC interface or by examining the database directly. In this Chapter, regardless of how one acquired the schema information, we assume that the schema information is encoded in a vector $\mathbb{R}$ defined below.

Let us assume the existence of a set $\widehat{T}$ of table names, a set $\widehat{C}$ of column names and a set $\widehat{b}$ of atomic base types defined in the standard SQL (e.g., integer, char, string). When name collision occurs, a column name $c \in \widehat{C}$ is qualified by a table name $t \in \widehat{T}$ using the "[ ]" notation (e.g., $t[c]$).

Definition 12 (Relational Schema) *A* relational schema *is denoted by 4-tuple* $\mathbb{R} = (T, C, P, \Delta)$, *where:*

- T *is a finite set of table names in* $\widehat{T}$,
- C *is a function from a table name* $t \in T$ *to a set of column names* $c \in \widehat{C}$,
- P *is a function from a column name* c *to its* column type definition*: i.e.,* $P(c) = \alpha$, *where* α *is a 5-tuple* (τ, u, n, d, f), *where* $\tau \in \widehat{b}$, u *is either "*υ*" (unique) or "*$\neg\upsilon$*" (not unique),* n *is either "?" (nullable) or "*$\neg$*?" (not nullable),* d *is a finite set of valid domain values of* c *or* ϵ *if not known, and* f *is a default value of* c *or* ϵ *if not known, and*
- Δ *is a finite set of relational integrity constraints that can be either retrieved from databases directly or provided by human experts.* □

Example 9. Consider two tables `student(Sname, Advisor, Course)` and `professor(Pname, Office)` where keys are underlined, and `Advisor` is a foreign key referencing `Pname` column. The column `Office` is an integer type, while the rest of the columns are string types. Also `Office` may be null. When student's advisor has not yet been decided, professor "Prof. Smith" will be the initial advisor. Student can have many advisors and take zero or more courses. The corresponding relational schema and data fragment are given in Table 7.1. □

$$
\begin{aligned}
T &= \{student, professor\} \\
C(student) &= \{Sname, Advisor, Course\} \\
C(professor) &= \{Pname, Office\} \\
P(Sname) &= (string, \neg v, \neg ?, \epsilon, \epsilon) \\
P(Advisor) &= (string, \neg v, \neg ?, \epsilon, \text{“}Prof.Smith'') \\
P(Course) &= (string, \neg v, \neg ?, \epsilon, \epsilon) \\
P(Pname) &= (string, v, \neg ?, \epsilon, \epsilon) \\
P(Office) &= (integer, \neg v, ?, \epsilon, \epsilon)
\end{aligned}
$$

$$
\Delta = \{\{Sname, Advisor, Course\} \overset{key}{\to} student, \; Pname \overset{key}{\to} professor, \; Advisor \subseteq Pname
$$

student		
`Sname`	`Advisor`	`Course`
John	Prof. Muntz	Multimedia
John	Prof. Zaniolo	Logic
John	Prof. Zaniolo	Data Mining
Tom	Prof. Muntz	Queueing Theory
Tom	Prof. Chu	Database Systems
Tom	Prof. Chu	Distributed Databases

professor	
`Pname`	`Office`
Prof. Muntz	600
Prof. Chu	550
Prof. Zaniolo	_
Prof. Parker	490

Table 7.1: Example of relational schema and data.

Next, let us define the output model. Lately, there have been about a dozen competing XML schema language proposals. Although XML-Schema is being shaped by W3C and will replace DTD soon, it is likely that different applications will choose different XML schema languages that best suit their particular purposes. Therefore, instead of choosing one language proposal, we formalize a core set of important features into a new notion of *XSchema* and use it as our output modeling language. The benefits of such formalization is that it is both concise and precise. More importantly, it breaks the tie between the translation algorithm that we are developing and the final schema language notations. Informally, *XSchema* borrows structural features from DTD and RELAX-NG, and data types and constraint specification features from XML-Schema. From a formal language and database perspective [MLM01c], *XSchema* is a local tree grammar extended with attribute, datatype and constraint specifications.

Starting from the notations in [FS00], we define *XSchema* below. We first assume the existence of a set $\widehat{E}$ of element names, a set $\widehat{A}$ of attribute names and a set $\widehat{\tau}$ of atomic data types defined in [BM01] (e.g., ID, IDREF, string, integer, date, etc). When needed, an attribute name $a \in \widehat{A}$ is qualified by the element names using the *path expression* notation $e_1.e_2 \cdots e_n.a$, where $e_i \in \widehat{E}, 1 \leq i \leq n$).

Definition 13 (*XSchema*) *An XSchema is denoted by 6-tuple* $\mathbb{X} = (E, A, M, P, r, \Sigma)$, *where:*

- *E is a finite set of element names in $\widehat{E}$,*
- *A is a function from an element name $e \in E$ to a set of attribute names $a \in \widehat{A}$,*
- *M is a function from an element name $e \in E$ to its* element type definition*: i.e., $M(e) = \alpha$, where α is a regular expression: $\alpha ::= \epsilon \mid \tau \mid \alpha + \alpha \mid \alpha, \alpha \mid$*

$\alpha^? \mid \alpha^* \mid \alpha^+$, where ϵ denotes the empty element, $\tau \in \widehat{\tau}$, "+" for the union, "," for the concatenation, "$\alpha^?$" for zero or one occurrence, "α^*" for the Kleene star, and "α^+" for "α, α^*",

- *P is a function from an attribute name a to its* attribute type definition*: i.e., $P(a) = \beta$, where β is a 4-tuple (τ, n, d, f), where $\tau \in \widehat{\tau}$, n is either "?" (nullable) or "¬?" (not nullable), d is a finite set of valid domain values of a or ϵ if not known, and f is a default value of a or ϵ if not known,*
- *$r \subseteq E$ is a finite set of root elements, and*
- *Σ is a finite set of integrity constraints for XML model* □

Translation from *XSchema* to the actual XML schema language notations is relatively straightforward and not discussed further in this Chapter. It is worthwhile to note, however, that depending on the chosen XML schema language, some of the features specifiable in *XSchema* might not be translatable at the end. For instance, any "non-trivial type" or composite key information would be lost if one decides to use DTD as the final XML schema language.

7.4 Flat Translation and Nesting-based Translation

XML model uses two basic building blocks to construct XML documents – attribute and element. A few basic characteristics inherited from XML model include: (1) the attributes of a node are not ordered, while the child elements of a node are ordered, (2) both support data types as specified in [BM01], and (3) elements can express multiple occurrences better than attributes. The detailed capabilities of those, however, vary depending on the chosen XML schema language. In translating $\mathbb{R}$ to $\mathbb{X}$, therefore, one can either use attribute or element in $\mathbb{X}$ to represent the same entity in $\mathbb{R}$ (e.g, a column with string type in $\mathbb{R}$ can be translated to either attribute or element with string type in $\mathbb{X}$).

To increase the flexibility of the algorithms, we assume that there are two modes – **attribute-oriented** and **element-oriented**. Depending on the mode, an algorithm can selectively translate an entity in $\mathbb{R}$ to either attribute or element if both can capture the entity correctly. However, if the chosen XML schema language requires attribute or element for an entity (e.g., a key column in $\mathbb{R}$ needs to be translated to an attribute with type `ID` in $\mathbb{X}$), we assume that the algorithm follows the limitations.

7.4.1 Flat Translation

The simplest translation method is to translate (1) tables in $\mathbb{R}$ to elements in $\mathbb{X}$ and (2) columns in $\mathbb{R}$ to attributes (in attribute-oriented mode) or elements (in element-oriented mode) in $\mathbb{X}$. These two modes are analogous except that element-oriented mode adds additional order semantics to the resulting schema. Since $\mathbb{X}$ represents the "flat" relational tuples faithfully, this method is called *Flat Translation* (FT). The general procedure of the **Flat Translation** is omitted in the interest of space and can be found in [LMC01]. One example of FT is shown below:

Example 10. $\mathbb{R}_9$ in Example 9 would be translated to $\mathbb{X}_{10} = (E, A, M, P, r, \Sigma)$ via FT, where

$$\begin{aligned}
E &= \{student, professor\} \\
A(student) &= \{Sname, Advisor, Course\} \\
A(professor) &= \{Pname, Age\} \\
M(student) &= \epsilon \\
M(professor) &= \epsilon
\end{aligned}$$

$$\begin{aligned}
P(Sname) &= (string, ?, \epsilon, \epsilon) \\
P(Advisor) &= (IDREF, ?, \epsilon, \text{``}J.Smith'') \\
P(Course) &= (string, ?, \epsilon, \epsilon) \\
P(Pname) &= (ID, \neg ?, \epsilon, \epsilon) \\
P(Age) &= (integer, ?, \epsilon, \epsilon) \\
r &= \{student, professor\}
\end{aligned}$$

$$\begin{aligned}
\Sigma = \{&\{Sname, Advisor, Course\} \overset{key}{\rightarrow} student, \\
&Pname \overset{key}{\rightarrow} professor, Advisor \subseteq Pname\} \qquad \square
\end{aligned}$$

FT is a simple and effective translation algorithm, but it has some problems. As the name implies, FT translates the "flat" relational model to a "flat" XML model in a one-to-one manner. The drawback of FT is that it does not utilize several basic "non-flat" features provided by XML for data modeling such as representing *repeating sub-elements* through regular expression operators (e.g., "*", "+"). We remedy this problem in the NeT algorithm below.

7.4.2 Nesting-based Translation

To remedy the problems of FT, one needs to utilize various *element content models* of XML. Towards this goal, we propose to use the *nest* operator [JS82]. Our idea is to find a "best" element content model that uses α^* or α^+ using the *nest* operator. First, let us define the *nest* operator. Informally, for a table t with a set of columns C, *nesting* on a non-empty column $X \in C$ collects all tuples that agree on the remaining columns $C - X$ into a set[4]. Formally,

Definition 14 (Nest) *[JS82]. Let t be a n-ary table with column set C. Let further $X \in C$ and $\overline{X} = C - X$. For each $(n-1)$-tuple $\gamma \in \Pi_{\overline{X}}(t)$, we define an*

[4]Here, we only consider single attribute nesting.

n-tuple γ^* *as follows:*

$$\left.\begin{array}{rcl}\gamma^*[\overline{X}] & = & \gamma \\ \gamma^*[X] & = & \{\kappa[X] \mid \kappa \in t \wedge \kappa[\overline{X}] = \gamma\}\end{array}\right\} \quad \textit{then}, \quad \textit{nest}_X(t) = \{\gamma^* \mid \gamma \in \Pi_{\overline{X}}(t)\}$$

□

After $nest_X(t)$, if column X has only a set with "single" value $\{v\}$ for all the tuples, then we say that **nesting failed** and we treat $\{v\}$ and v interchangeably (i.e., $\{v\} = v$). Thus when nesting failed, the following is true: $nest_X(t) = t$. Otherwise, if column X has a set with "multiple" values $\{v_1, ..., v_k\}$ with $k \geq 2$ for at least one tuple, then we say that **nesting succeeded**. The general procedure for nesting is given in Table 7.2.

Example 11. Consider a table R in Table 7.3, where column names containing a set after nesting (i.e., nesting succeeded) are appended by "+" symbol. Here we assume that the columns A, B, C are non-nullable. In computing $nest_A(R)$ at (b), the first, third, and fourth tuples of R agree on their values in columns (B, C) as (a, 10), while their values of the column A are all different. Therefore, these different values are grouped (i.e., nested) into a set {1,2,3}. The result is the first tuple of the table $nest_A(R)$ – ({1,2,3}, a, 10). Similarly, since the sixth and seventh tuples of R agree on their values as (b, 20), they are grouped to a set {4,5}. In computing $nest_B(R)$ at (c), there are no tuples in R that agree on the values of the columns (A, C). Therefore, $nest_B(R) = R$. In computing $nest_C(R)$ at (d), since the first two tuples of R – (1, a, 10) and (1, a, 20) – agree on the values of the columns (A, B), they are grouped to (1, a, {10,20}). Nested tables (e) through (j) are constructed similarly. □

Since the *nest* operator requires scanning of the entire set of tuples in a given table, it can be quite expensive. In addition, as shown in Example 11, there

NeT: $\mathbb{R} = (T, C, P, \Delta) \Longrightarrow \mathbb{X} = (E, A, M, P, r, \Sigma)$

1. Each table t_i in $\mathbb{R}$ is translated to an element e_i in $\mathbb{X}$: $E = \bigcup_{\forall i}\{e_i\}$.

2. For each table t_i in $\mathbb{R}$, apply the *nest* operator repeatedly until no nesting succeeds. Choose the best nested table based on the selected criteria. Denote this table as $t'_i(c_1, \ldots, c_{k-1}, c_k, \ldots, c_n)$, where nesting succeeded on the columns $\{c_1, \ldots, c_{k-1}\}$. If $k = 1$ (i.e., no nesting succeeded), follow the flat translation. Otherwise, do the following:

   (a) For each column c_i ($1 \leq i \leq k-1$), where $P(c_i) = (\tau, u, n, d, f)$, if $n = ?$, then the content model is $M(e_i) = (\ldots, c_i^*, \ldots)$, otherwise $M(e_i) = (\ldots, c_i^+, \ldots)$.

   (b) For each column c_j ($k \leq j \leq n$), where $P(c_j) = (\tau, u, n, d, f)$, do flat translation

   - (element-oriented mode) if $n = ?$, the content model is $M(e_i) = (\ldots, c_j^?, \ldots)$, otherwise $M(e_i) = (\ldots, c_j, \ldots)$.
   - (attribute-oriented mode) if c_j is translated to a_j, then $A(e_i) = \bigcup_{\forall j}\{a_j\}$ and $P(a_j) = (\tau,\ n,\ d,\ f)$.

3. All elements e_i in $\mathbb{X}$ become roots: $r = \bigcup_{\forall i}\{e_i\}$.

4. Copy Δ in $\mathbb{R}$ into Σ in $\mathbb{X}$.

Table 7.2: NeT algorithm.

	A	B	C
#1	1	a	10
#2	1	a	20
#3	2	a	10
#4	3	a	10
#5	4	b	10
#6	4	b	20
#7	5	b	20

(a) R

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(b) $nest_A(R)$

A	B	C
1	a	10
1	a	20
2	a	10
3	a	10
4	b	10
4	b	20
5	b	20

(c) $nest_B(R) = R$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(d) $nest_C(R)$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(e) $nest_B(nest_A(R))$
$= nest_C(nest_A(R))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(f) $nest_A(nest_C(R))$

A	B	C^+
1	a	{10,20}
2	a	10
3	a	10
4	b	{10,20}
5	b	20

(g) $nest_B(nest_C(R))$

A^+	B	C
{1,2,3}	a	10
1	a	20
4	b	10
{4,5}	b	20

(h) $nest_C(nest_B(nest_A(R)))$
$= nest_B(nest_C(nest_A(R)))$

A^+	B	C^+
1	a	{10,20}
{2,3}	a	10
4	b	{10,20}
5	b	20

(i) $nest_B(nest_A(nest_C(R)))$
$= nest_A(nest_B(nest_C(R)))$

Table 7.3: A relational table R and its various nested forms.

are various ways to nest the given table. Therefore, it is important to find an efficient way (that uses the *nest* operator minimum number of times) of obtaining an acceptable element content model.

First, to find out the total number of ways to nest, let us use the following two properties [JS82]:

$$\begin{aligned} P1: \quad nest_A(nest_B(t)) &\neq nest_B(nest_A(t)) \\ P2: \quad nest_X(nestAll_L(t)) &= nestAll_L(t), \text{ if } X \in L. \end{aligned}$$

Here, $nestAll_L(t)$ represents performing nesting on the columns on L in the order as shown below: $nestAll_{L=<c_1,c_2,\ldots,c_n>}(t) = nest_{c_1}(nest_{c_2}(\ldots(nest_{c_n}(t))))$ P1 states that "commutativity" of nesting does not hold in general and P2 states that nesting along the same column repeatedly has the property of "idempotency". Using the two properties, the number of permutations to nest tables can be described as follows:

Remark 1 *Using the* falling factorial power *notation "x to the m falling" as $x^{\underline{m}}$ in [GKP94], the total number of different nestings T for a table with n columns is given by:* $T = \sum_{k=1}^{n} n^{\underline{k}}$ □

Proof. The number of the *first* nesting along n columns is the same as the number of 1-element sequences: n. The number of the *second* nesting along n columns is again the same as the number of 2-elements sequences by P2: $n(n-1)$. Continuing this, the number of the *last* nesting along n columns is again the same as the number of n-elements sequences: $n + n(n-1) + \cdots + n(n-1)\ldots(2)(1) = n^{\underline{1}} + n^{\underline{2}} + \cdots + n^{\underline{n}} = \sum_{k=1}^{n} n^{\underline{k}}$ (q.e.d)

According to Remark 1, there are 15 meaningful ways of nesting along the columns A, B, C in Table 7.3. Then, the next questions are (1) how to decrease

T by avoiding unnecessary nesting, and (2) which nesting should be chosen as *the* translation. To answer these questions, let us first describe a few useful properties of the *nest* operator as follows:

Lemma 4. *Consider a table t with column set C, and candidate keys, $K_1, K_2, \ldots,$ $K_n \subseteq C$. Applying the nest operator on a column $X \notin (K_1 \cap K_2 \cap \ldots \cap K_n)$ yields no changes.* ■

PROOF. Consider a column $X \in C$, such that X is not an attribute of at least one of the candidate keys, say $X \notin K_i$. Now $\overline{X} \supseteq K_i$, and hence $\overline{X}$ is unique. Thus, no two tuples can agree on $\overline{X}$. Therefore, by the definition of the *nest* operator, nesting on X will fail. (q.e.d)

COROLLARY 2. For any nested table $nest_X(t)$, $\overline{X} \rightarrow X$ holds. (q.e.d)

Corollary 2 states that after applying the nest operator of column X, the remaining columns $\overline{X}$ become a super key. Fischer et al. [FST85] have proved that functional dependencies are preserved against nesting as follows:

Lemma 5. *[FST85] If X, Y, Z are columns of t, then: $t : X \rightarrow Y \Longrightarrow nest_Z(t) :$ $X \rightarrow Y$* ■

Now, we arrive at the following useful property:

Theorem 1. *Consider a table t with column set C, candidate keys, $K_1, K_2, \ldots,$ $K_n \subseteq C$, and column set K such that $K = K_1 \cap K_2 \cap \ldots \cap K_n$. Further, let $|C| = n$ and $|K| = m$ $(n \geq m)$. Then, the number of necessary nestings, N, is bounded by $N \leq \sum_{k=1}^{m} m^{\underline{k}}$* ■

PROOF. The first column to be nested, say X, is chosen such that $X \in K$ by Lemma 4, in one of the m ways. Now after the first nesting, by Corollary 2, we have a new candidate key $\overline{X}$. The next column to be nested is chosen from $K \cap \overline{X}$, where $|K \cap \overline{X}| = m - 1$. Thus we have $m - 1$ ways of chosing the second column for nesting. Continuing this, we have total number of nesting is $m + m(m-1) + \ldots + m(m-1)\ldots(2)(1) = \sum_{k=1}^{m} m^{\underline{k}}$. (q.e.d)

Note that in general m is much smaller than n in Theorem 1, thus reducing the number of necessary nesting significantly.

Example 12. Consider a table R in Table 7.3 again. Suppose attributes A and C constitute a key for R. Since nesting on the same column repeatedly is not useful by property P2 there is no need to construct, for instance, $nest_A(nest_A(R))$. Since nesting on a non-key column is not useful by Lemma 4, nesting along column B (e.g., $nest_B(R)$ at (c)) can be avoided. Furthermore, the functional dependency (i.e., $AC \overset{key}{\to} R = AC \to \overline{AC} = AC \to B$) persists after nesting on either column A or C by Lemma 5. Consequently, one needs to construct only the following nested tables: $nest_A(R)$ at (b), $nest_C(R)$ at (d), $nest_C(nest_A(R))$ at (e), $nest_A(nest_C(R))$ at (f). □

As we have shown, when candidate key information is available, the number of nestings to be performed can be reduced. However, when such information is not known, the *nest* operator must be applied for all possible combinations in Remark 1. After applying the *nest* operator to the given table repeatedly, there can be still several nested tables where nesting succeeded. In general, the choice of the final schema should take into consideration the semantics and usages of the underlying data or application and this is where user intervention is beneficial. By default, without further input from users, NeT chooses as the final schema the nested table where the most number of nestings succeeded - this is a schema which provides low "data redundancy" - as given in Table 7.2.

Example 13. Using NeT with the element-oriented mode, $\mathbb{R}_9$ in Example 9 would be translated to $\mathbb{X}_{13} = (E, A, M, P, r, \Sigma)$, where

$$\begin{aligned}
E &= \{student, professor\} \\
A(professor) &= \{Pname\} \\
M(student) &= (Sname, Advisor^{+}, Course^{+}) \\
M(professor) &= (Age^{?}) \\
P(Pname) &= (ID, \neg ?, \epsilon, \epsilon) \\
r &= \{student, professor\}
\end{aligned}$$

$$\begin{aligned}
\Sigma = \{ & \{Sname, Advisor, Course\} \overset{key}{\rightarrow} student, \\
& Pname \overset{key}{\rightarrow} professor, Advisor \subseteq Pname\}
\end{aligned}$$

$\mathbb{X}_{13}$ can be further rewritten in XML-Schema notations as shown in Table 7.4. Note that all three semantic constraints in Σ are captured in XML-Schema due to its sufficient expressive power. $\square$

We expect that the NeT algorithm will be especially useful in two scenarios, outlined below.

- The given relation is in 3NF (or BCNF) but not in 4NF. Non-fully normalized relations occur quite commonly in legacy databases, and they exhibit data redundancy. The NeT algorithm helps to decrease the data redundancy in such cases.

  As an example, consider the relation `ctx(Course, Teacher, Text)`, which gives the set of teachers and the set of text books for each course. Assume that the following multivalued dependencies hold, `Course` $\twoheadrightarrow$ `Teacher`, and `Course`$\twoheadrightarrow$`Text`. Suppose the relation *ctx* is represented as such (i.e., `ctx` is not in 4NF). The key for this relation is given by {`Course, Teacher,`

```
<schema>
  <element name="student">
   <complexType><sequence>
     <element ref="Sname"/>
     <element ref="Gender" minOccurs="0" maxOccurs="1"/>
     <element ref="Advisor" maxOccurs="unbounded"/>
     <element ref="Course" minOccurs="0" maxOccurs="unbounded"/>
   </sequence></complexType>
  </element>
  <element name="professor">
   <complexType>
    <sequence> <element ref="Age" minOccurs="0" maxOccurs="1"/>
    </sequence>
    <attribute name="Pname" type="string" use="required"/>
   </complexType>
  </element>
  <element name="Sname"> <complexType mixed="true"/> </element>
  <element name="Gender"> <complexType mixed="true"/> </element>
  <element name="Advisor"> <complexType mixed="true"/> </element>
  <element name="Course"> <complexType mixed="true"/> </element>
  <element name="Age"> <complexType mixed="true"/> </element>
  <key name="ekey">
   <selector xpath="//student"/>
    <field xpath="Sname"/> <field xpath="Advisor"/>
    <field xpath="Course"/>
  </key>
  <key name="pkey"> <selector xpath="//professor"/>
    <field xpath="@Pname"/> </key>
  <keyref refer="pkey"> <selector xpath="//student"/>
    <field xpath="Advisor"/> </key>
</schema>
```

Table 7.4: An XML schema equivalent to a relational schema of Example 9 in XML-Schema notations.

Text} $\overset{key}{\rightarrow}$ctx. When we do nesting on ctx, we will get the following table ctx'(Course, Teacher$^+$, Text$^+$). Thus NeT helps in removing data redundancies arising from multivalued dependencies.

- It is sometimes possible to represent the given relation "more intuitively" as a nested table by performing grouping on one or more of the attributes. As an example, consider the relation emp(empNum, branch) where the key is given by empNum $\overset{key}{\rightarrow}$ emp. This relation gives the employees and the branch where they work. When NeT is applied on the above relation, we might get the new nested relation as emp'(empNum$^+$, branch). This relation has grouped the list of employees by their branch.

Thus we observe that NeT is useful for decreasing data redundancy and obtaining a "more intuitive" schema by (1) removing redundancies caused by multivalued dependencies and (2) performing grouping on attributes. However NeT considers tables one by one, and *cannot* obtain a *big picture* of the relational schema where many tables are interconnected with each other through various other dependencies such as inclusion dependencies. To remedy this problem, we propose to use other semantic constraints of relational schema.

7.5 Translation using Inclusion Dependencies

In this section, we consider one kind of semantic constraints called *Inclusion Dependency (IND)* in database theory. Other kinds of semantic constraints such as *Functional Dependency (FD)* or *Multi-Valued Dependency (MVD)* are not considered in this section for the following reasons: In general, most CASE tools for relational database design generate schemas at least in 3NF. If there existed any MVDs, for instance, then the table would have been split up (i.e., normalized to 4NF) to avoid excessive data redundancy. Even if it was not normalized to 4NF, if we apply the NeT algorithm, most of the MVDs would have been removed.

General forms of INDs are difficult to acquire from the database automatically. However, we shall consider the most pervasive form of INDs - foreign key constraints - which can be queried through ODBC/JDBC interface. We study the translation of inclusion dependencies incrementally in three steps. In the first step, we consider the simplest case - one foreign key constraint defined between two tables. In the second step, we consider the case when there exist two foreign key constraints among three tables. In the third step, we consider the general case of mapping any given relational schema.

7.5.1 One Foreign Key between two Tables

Foreign key constraints are a special kind of INDs where the attributes being referenced form the *primary key* of the referenced relation. For two distinct tables s and t with lists of columns X and Y, respectively, suppose we have a foreign key constraint $s[\alpha] \subseteq t[\beta]$, where $\alpha \subseteq X$ and $\beta \subseteq Y$. Also suppose that $K_s \subseteq X$ is the key for s. Then, rewriting this in $\mathbb{R}$ notation, we have: $T = \{s, t\}, C(s) = \{X\}, C(t) = \{Y\}, \Delta = \{s[\alpha] \subseteq t[\beta], \beta \overset{key}{\rightarrow} t, K_s \overset{key}{\rightarrow} s\}$.

Different cardinality binary relationships between s and t can be expressed in the relational model by a combination of the following: (1) α is unique/not-unique (2) α is nullable/non-nullable.

The translation of two tables s, t with a foreign key constraint into *XSchema*, summarized in Table 7.5, works as follows:

- If α is non-nullable (i.e., none of the columns of α can take null values), then:

  - If α is unique, then there is a $1:1$ relationship between s and t. This can be captured as a sub-element $M(t) = (Y, s?)$.
  - If α is not-unique, then there is a $1:n$ relationship between s and t, and this is captured as a sub-element $M(t) = (Y, s^*)$.

  If s is represented as a sub-element of t, then the key for s will change from K_s to $(K_s - \alpha)$. The key for t will remain the same.

- If α is nullable, then the IND is represented as such in *XSchema*. Here we do flat translation on s, and copy the IND $s[\alpha] \subseteq t[\beta]$ to Σ.

Let us study the case when α is nullable more closely with the following example. Consider the relation $t(w_1, w_2, w_3)$ with key (w_1, w_2). Let t have the following tuples: $\{(1, 1, 1)\}$. Now consider $s(v_1, v_2, v_3)$ with key (v_2, v_3), and IND $s[v_1, v_2] \subseteq t[w_1, w_2]$. Let s have the following tuples: $\{(null, 1, 1), (null, 1, 2), (null, 2, 1), (1, 1, 3)\}$. We can observe that we *cannot* represent s as $s(v_3)$, and obtain the values of (v_1, v_2) for an s tuple by representing this s tuple as a child of a t tuple, or by having an IDREF attribute for the s tuple that refers to a t tuple. This is because v_1 is nullable.

In such a case, we represent the IND as such in *XSchema*. In this Chapter, we are concerned mostly with the usage of sub-elements and IDREF attribute for translation, and therefore, we will focus on the case when α is non-nullable, unless stated otherwise.

Example 14. Consider two tables `student` and `professor` of Example 9 again. There is a foreign key *Advisor* $\subseteq$ *Pname* and *Advisor* is not unique. Using the above rules, the schema will be mapped to the following XML schema in DTD notation:

α	$s:t$	*XSchema*
υ, $\neg?$	$(1,1):(0,1)$	$M(t)=(Y,s?)$, $M(s)=(X-\alpha)$, $\Sigma=\{(K_s-\alpha)\overset{key}{\to}s,\beta\overset{key}{\to}t\}$
υ, $?$	$(0,1):(0,1)$	$M(t)=(Y)$, $M(s)=(X)$, $\Sigma=\{s[\alpha]\subseteq t[\beta],\ K_s\overset{key}{\to}s,\beta\overset{key}{\to}t\}$
$\neg\upsilon$, $\neg?$	$(1,1):(0,n)$	$M(t)=(Y,s^*)$, $M(s)=(X-\alpha)$, $\Sigma=\{(K_s-\alpha)\overset{key}{\to}s,\beta\overset{key}{\to}t\}$
$\neg\upsilon$, $?$	$(0,1):(0,n)$	$M(t)=(Y),M(s)=(X)$, $\Sigma=\{s[\alpha]\subseteq t[\beta],\ K_s\overset{key}{\to}s,\beta\overset{key}{\to}t\}$

Table 7.5: Different values taken by α, the corresponding cardinality of the binary relationship between s and t, and the corresponding translation to *XSchema*.

```
<!ELEMENT professor  (Pname,Age,student*)>
<!ELEMENT student    (Sname,Course)>
```

Note the usage of `*` attached to the sub-element *student*. Note further that to identify a unique `student` element for a given professor, one needs now only *Sname* and *Course* pair (*Advisor* attribute was removed from the original key attribute list). □

7.5.2 Two Foreign Key among three Tables

Now consider the case where two foreign key constraints exist among three tables s, t_1, t_2 with a list of columns X, Y_1, Y_2, respectively, such that $s[\alpha]\subseteq t_1[\beta_1]$ and $s[\gamma]\subseteq t_2[\beta_2]$, where $\alpha,\gamma\subseteq X$ and are non-nullable, $\beta_1\subseteq Y_1$ and $\beta_2\subseteq Y_2$. If one applies the mapping rules for the case of a foreign key between two tables in Section 7.5.2 one at a time, then one will have a combination of the following depending on whether α and γ are unique or not: (1) $M(t_1)=(Y_1,s?)$ or $M(t_1)=$

(Y_1, s^*), (2) $M(t_2) = (Y_2, s?)$ or $M(t_2) = (Y_2, s^*)$.

The above translation has redundancy, and it exhibits the phenomenon known in database theory as "update anomaly" for s. That is, when one wants to update data for s, he/she needs to update s in two different places – fragment of s data under both t_1 and t_2. On the contrary, the original relational schema is "better" because one needs to update tuples of s in a single place. The same problem occurs for the case of "delete" as well. To avoid these anomalies, one of the two foreign key constraints should be captured either using INDs or using IDREF attributes. For example, let us assume that the first foreign key constraint $s[\alpha] \subseteq t_1[\beta_1]$ is represented as $M(t_1) = (Y_1, s^*)$, $M(s) = (X - \alpha)$. Then the second foreign key constraint $s[\gamma] \subseteq t_2[\beta_2]$ can be represented using IDREF attribute as follows:

$$\begin{aligned} A(t_2) &= \{ID\_t_2\}, & P(ID\_t_2) &= (ID, \neg?, \epsilon, \epsilon) \\ A(s) &= \{Ref\_t_2\}, & P(Ref\_t_2) &= (IDREF, \neg?, \epsilon, \epsilon) \\ M(t_2) &= (Y_2), & M(s) &= (X - \alpha - \gamma) \end{aligned}$$

Let us denote the old and new keys for s as K_s and K'_s, respectively. Then, K'_s is determined as follows: (1) if $\alpha \cap K_s = \phi$, then $K'_s = K_s$, and (2) if $\alpha \cap K_s \neq \phi$, then $K'_s = (K_s - \alpha) \cup Ref\_t_2$

Example 15. In addition to two tables `student` and `professor` of Example 14, consider a third table `class(`<u>`Cname`</u>`, Room)` with a second foreign key $student[Course] \subseteq class[Cname]$. Then, using the above rules, the schema will be mapped to the following XML schema in DTD notation:

```
<!ELEMENT professor  (Pname,Age,student*)>
<!ELEMENT student    (Sname)>
<!ELEMENT class      (Cname,Room)>
```

```
<!ATTLIST student     Ref_class IDREF>
<!ATTLIST class       ID_class ID>
```

Note the addition of two new attributes - `Ref_class` of type $IDREF$ and `ID_class` of type ID. The new key for *student* is given by $\{Sname, Ref\_class \overset{key}{\rightarrow} student\}$, which cannot be represented in DTD. □

Note that between two foreign keys, deciding which one is represented as sub-element and which one is represented as IDREF attribute can best be done based on further semantics.

7.5.3 A General Relational Schema

Now let us consider the most general case with set of tables $\{t_1, ..., t_n\}$ and INDs $t_i[\alpha_i] \subseteq t_j[\beta_j]$, where $i, j \leq n$. We consider only those INDs that are foreign key constraints (i.e., $\beta_j \overset{key}{\rightarrow} t_j$), and where α_i is non-nullable. The relationships among tables can be captured by a graph representation, termed as IND-Graph.

Definition 15 (IND-Graph) *An* IND-Graph $G = (V, E)$ *consists of a node set V and a directed edge set E, such that for each table t_i, there exists a node in V, and for each distinct IND $t_i[\alpha] \subseteq t_j[\beta]$, $t_j \rightarrow t_i$ exists in G.* □

Note the edge direction is reversed from the IND direction for convenience. Given a set of INDs, such IND-Graph can be easily constructed. Once IND-Graph is constructed, one needs to decide the starting point to apply translation rules. For that purpose, we use the notion of **top nodes** similar to the one in [STH99, LC00b], where an element is a top node if it *cannot* be represented as a sub-element of any other element. Such top nodes can be identified as follows:

1. An element s is a top node, if there exists no other element t, $t \neq s$, where there is a IND of the form $s[\alpha] \subseteq t[\beta]$, and α is non-nullable.

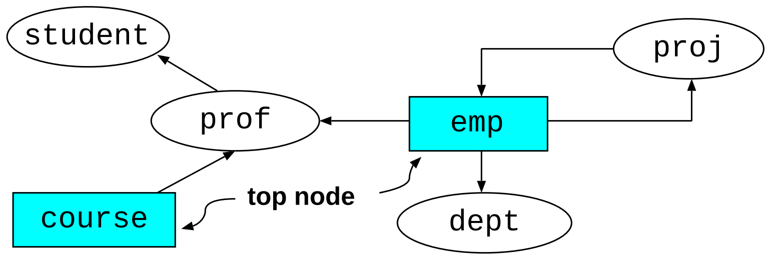


Figure 7.2: The IND-Graph representation of the schema of Table 7.7.

2. Consider a set of elements $S = s_1, s_2, \ldots, s_k$ that form a cyclic set of INDs and none of the elements in S is a top node by 1. Suppose there exists no element $t \notin S$, such that there is a IND of the form $s_j[\alpha] \subseteq t[\beta]$, and α is non-nullable. In this case, choose any one of the elements in S as a top node.

Let T denote the set of top nodes. After identifying the top nodes, we traverse G, using say Breadth-First Search (BFS), until we traverse all the nodes and edges, and represent the INDs as sub-elements or IDREF attributes. The algorithm for **Constraint-based Translation** (CoT) is given in Table 7.6.

Example 16. Consider a schema and its associated INDs in Table 7.7. The IND-Graph is shown in Figure 7.2. Two top nodes are identified (1) `course`: There is no node t, where there is an IND of the form `course` $[\alpha] \subseteq t[\beta]$, and (2) `emp`: There is a cyclic set of INDs between `emp` and `proj`, and there exists no node t such that there is an IND of the form `emp` $[\alpha] \subseteq t[\beta]$ or `proj` $[\alpha] \subseteq t[\beta]$. Therefore of `emp` and `proj` we decided to choose `emp` arbitrarily. Following list shows one of the possible orders in which the different INDs are visited, the choice made to represent the IND (either sub-element or IDREF attribute), and the resulting changes in *XSchema*.

1. `prof(Teach)` $\subseteq$ `course(Cid)`: M (`course`) = (`Cid`, `Title`, `Room`, `prof`*),

```
CoT: $\mathbb{R} = (T, C, P, \Delta) \Longrightarrow \mathbb{X} = (E, A, M, P, r, \Sigma)$

  1. Construct IND-Graph $G = (V, E)$ from the given INDs; Identify $T$, the set of top
     nodes. Define $S = T$ to keep track of top-nodes and nodes that are represented
     as sub-elements.

  2. For each top-node $t \in T$, do BFS. Suppose we reach a node $w$ from $v$ (i.e., IND:
     $w[\alpha] \subseteq v[\beta]$); Let $C(w) = C_w$, and $C(v) = C_v$.

     (a) If $w \notin S$ (i.e., $w$ is *not* yet a sub-element of some other node), translate the
         IND as in Section 7.5.1.

           i. If $\alpha$ is unique, then $M(v) = (C_v, w?)$.
          ii. If $\alpha$ is not-unique, then $M(v) = (C_v, w^*)$.
         iii. $M(w) = (C_w - \alpha)$.
          iv. $S = S \ \cup \ w$.

     (b) If $w \in S$ (i.e., $w$ is already a sub-element of some other node), translate the
         IND as IDREF attribute as in Section 7.5.2.

           i. $A(v) = \{ID\_v\}$, $A(w) = \{Ref\_v\}$, $M(v) = (C_v)$, $M(w) = (C_w - \alpha)$,
              $\Sigma = K'_w \overset{key}{\rightarrow} w$.

  3. Copy the remaining integrity constraints in $\Delta$ to $\Sigma$. Also set $r = T$.
```

Table 7.6: CoT algorithm.

student(Sid, Name, Advisor)
emp(Eid, Name, ProjName)
prof(Eid, Name, Teach)
course(Cid, Title, Room)
dept(Dno, Mgr)
proj(Pname, Pmgr)
student(Advisor) $\subseteq$ prof(Eid)
emp(ProjName) $\subseteq$ proj(Pname)
prof(Teach) $\subseteq$ course(Cid)
prof(Eid, Name) $\subseteq$ emp(Eid, Name)
dept(Mgr) $\subseteq$ emp(Eid)
proj(Pmgr) $\subseteq$ emp(Eid)

Table 7.7: Example of a relational schema with complex INDs.

M (prof) = (Eid, Name)

2. student(Advisor) $\subseteq$ prof(Eid): M(prof) = (Eid, Name, student*), M(student) = (Sid, Name)

3. dept(Mgr) $\subseteq$ emp(Eid): M(emp) = (Eid, Name, ProjName, dept*), M(dept) = (Dno)

4. proj(Pmgr) $\subseteq$ emp(Eid): M(emp) = (Eid, Name, ProjName, dept*, proj*), M(proj) = (Pname)

5. emp(ProjName) $\subseteq$ proj(Pname): M(emp) = (Eid, Name, dept*, proj*), A(proj) = {ID_proj}, A(emp) = {Ref_proj}

6. prof(Eid, Name) $\subseteq$ emp(Eid, Name): M(prof) = (student*), A(emp) = {ID_emp}, A(prof) = {Ref_emp}, $\Sigma = \{\text{Ref\_emp} \overset{key}{\rightarrow} prof\}$ □

Example 17. $\mathbb{X}_{16}$ of Example 16 can be further rewritten in element-oriented mode to DTD notations as follows:

```
<!ELEMENT course  (Cid, Title, Room, prof*)>
<!ELEMENT prof    (Name, student*)>
<!ATTLIST prof     Eid      ID>
<!ELEMENT student (Sid, Name)>
<!ELEMENT emp     (Eid, Name, ProjName, dept*, proj*)>
<!ATTLIST emp      Ref_prof IDREF>
<!ELEMENT dept    (Dno)>
<!ELEMENT proj    (Pname)>
<!ATTLIST proj     Ref_emp IDREFS>
```

It is worthwhile to point out that there are several places in CoT where human experts can determine better mapping based on the semantics and usages of the underlying data or application.

- The CoT algorithm identifies a minimal set of top-nodes, breaking any ties (when there are cyclic INDs) arbitrarily. A better mapping might have more top-nodes than this minimal set, or might choose to break a tie in a particular manner.
- Given a set of foreign-key constraints on one table, CoT chooses one foreign-key constraint to be represented as a sub-element, and represents the remaining using IDREF attributes. Human experts might be able to provide better input as to which constraint should be represented as sub-element, and which as IDREF attributes.

Examples so far have shown the conversion flow of $\mathbb{X} \rightarrow$ CoT $\rightarrow$ DTD. We can also have the conversion flow $\mathbb{X} \rightarrow$ NeT $\rightarrow$ CoT $\rightarrow$ DTD. However this imposes a restriction; when NeT followed by CoT are applied, nesting can be done only on attributes that do not participate in any IND. One such example is shown below:

Example 18. Consider $\mathbb{R}_9$ in Example 9. Let us first apply NeT and then CoT on this. When we apply NeT, we perform nesting only on the column `Course` of `student`, and obtain a content model for `student` as $M(student) = (Sname, Advisor, Course^+)$. Now applying CoT on the above schema, we get the output *XSchema* as $\mathbb{X}_{18} = (E, A, M, P, r, \Sigma)$, where

$$\begin{aligned}
E &= \{student, professor\} \\
A(professor) &= \{Pname\} \\
M(student) &= (Sname, Course^+) \\
M(professor) &= (Age^?, student^*) \\
P(Pname) &= (ID, \neg ?, \epsilon, \epsilon) \\
r &= \{student, professor\} \\
\Sigma &= \{\{Sname, Course\} \overset{key}{\rightarrow} student, \\
&\quad Pname \overset{key}{\rightarrow} professor\} \qquad \Box
\end{aligned}$$

7.6 Discussion

All three algorithms – FT, NeT, and CoT – are "correct" in the sense that they all have preserved the original information of relational schema. For instance, using the notion of information capacity [MIR94], a theoretical analysis for the correctness of our translation procedures is possible; we can actually show that NeT and CoT algorithms are *equivalence preserving transformations*. However, we defer this detailed analysis to a later version.

With respect to the "goodness" of XML schema that the proposed algorithms generate, it is not obvious to bluntly state whether or not they are good, since there has not been any unanimous normalization theory for XML model yet. Some early work for nested relational model (e.g., [OY87]) is related, but more recently a few proposals have been made for normal forms of XML model (e.g., [EM01, WLL01]). To a greater or lesser extent, the crux of such normal forms is an attempt to reduce data redundancy so that various anomalies can be avoided. Although the output schema from NeT or CoT does not exactly fit into normal forms defined by [EM01, WLL01], they share similar properties. For instance, identifying multivalued attributes and making them repeating subelements in NeT is essentially a necessary step towards "object class normal form" in [WLL01]. The use of reference attributes in CoT for handling multiple foreign key constraints defined on one table (Section 7.5.2) can be explained similarly. Therefore, we would like to point out that although it is early to formally prove the goodness of our proposals, it is evident that our proposals lead to less *redundant* yet *correct* XML schema.

7.7 Experimental Results

7.7.1 NeT Results

We compare the results of FT and NeT with that of DB2XML v 1.3 [Tur99]. Consider the `Orders` table (containing 830 tuples) found in MS Access NorthWind sample database.

```
Orders (CustomerID, EmployeeID, ShipVia, ShipAddress,
        ShipCity, ShipCountry, ShipPostalCode)
```

Tables 7.8 and 7.9 show the DTDs generated by DB2XML, FT in attribute-oriented mode, NeT in both element-oriented and attribute-oriented modes, re-

```
<!ELEMENT Orders (CustomerID,EmployeeID,ShipVia,ShipAddress,
                  ShipCity,ShipCountry,ShipPostalCode)>
<!ELEMENT CustomerID      (#PCDATA)>
<!ATTLIST CustomerID      ISNULL (true|false) #IMPLIED>
<!ELEMENT EmployeeID      (#PCDATA)>
<!ATTLIST EmployeeID      ISNULL (true|false) #IMPLIED>
<!ELEMENT ShipVia         (#PCDATA)>
<!ATTLIST ShipVia         ISNULL (true|false) #IMPLIED>
...
<!ELEMENT ShipCountry     (#PCDATA)>
<!ATTLIST ShipCountry     ISNULL (true|false) #IMPLIED>
<!ELEMENT ShipPostalCode  (#PCDATA)>
<!ATTLIST ShipPostalCode  ISNULL (true|false) #IMPLIED>
```

(a) DB2XML

```
<!ELEMENT  Orders  (EMPTY)>
<!ATTLIST  Orders
           CustomerID     CDATA #IMPLIED
           EmployeeID     CDATA #IMPLIED
           ShipVia        CDATA #IMPLIED
           ShipAddress    CDATA #IMPLIED
           ShipCity       CDATA #IMPLIED
           ShipCountry    CDATA #IMPLIED
           ShipPostalCode CDATA #IMPLIED>
```

(b) FT in attribute-oriented mode

Table 7.8: DTDs generated by FT algorithm.

spectively. In Table 7.8(a), DB2XML always uses element to represent columns of a table. To represent whether the column is nullable or not, DB2XML adds a special attribute `ISNULL` to every element: i.e., "`ISNULL = true`" means the column is nullable. In Table 7.8(b), FT in attribute-oriented mode uses `#IMPLIED` or `REQUIRED` to represent whether the column is nullable or not. Observe that both DB2XML and FT share the same problem of translating "flat" relational schema to "flat" XML schema. In both Tables 7.9(a) and 7.9(b), NeT finds two columns `EmployeeID` and `ShipVia` can be nested. Intuitively, the new schema infers that

```
<!ELEMENT  Orders  (CustomerID,EmployeeID+,ShipVia*,ShipAddress?,
                    ShipCity?,ShipCountry?,ShipPostalCode?)>
<!ELEMENT  CustomerID     (#PCDATA)>
<!ELEMENT  EmployeeID     (#PCDATA)>
<!ELEMENT  ShipVia        (#PCDATA)>
<!ELEMENT  ShipAddress    (#PCDATA)>
<!ELEMENT  ShipCity       (#PCDATA)>
<!ELEMENT  ShipCountry    (#PCDATA)>
<!ELEMENT  ShipPostalCode (#PCDATA)>
```

(a) NeT in element-oriented mode

```
<!ELEMENT  Orders  (EmployeeID+,ShipVia*)>
<!ATTLIST  Orders
           CustomerID      CDATA #REQUIRED
           ShipAddress     CDATA #IMPLIED
           ShipCity        CDATA #IMPLIED
           ShipCountry     CDATA #IMPLIED
           ShipPostalCode  CDATA #IMPLIED>
<!ELEMENT  EmployeeID  (#PCDATA)>
<!ELEMENT  ShipVia     (#PCDATA)>
```

(b) NeT in attribute-oriented mode

Table 7.9: DTDs generated by NeT algorithm.

for each `CustomerID`, multiple non-zero `EmployeeID` and multiple `ShipVia` can exist. Also NeT finds that `CustomerID` is a mandatory column. To ensure this property, Table 7.9(a) adds no suffix such as `?` or `*` to `CustomerID` sub-element and Table 7.9(b) uses `#REQUIRED` construct explicitly. Not only DTDs found by NeT are more succinct than one by DB2XML, We observe that the DTDs found by NeT are more succinct and more intuitive than the ones found by DB2XML.

We implemented the NeT algorithm described in Table 7.2, where we used two additional optimization rules: (1) if $nest_X(t) = t$, then $nest_X(nestAll_L(t)) = nestAll_L(t)$ for any list of columns, L, and (2) if $nest_X(nestAll_L(t)) = nestAll_L(t)$ for any column X and all possible list of columns L of length l, then $nest_X(nestAll_M(t)) =$

Test Set	# of attr. / tuple	NestRatio	ValueRatio	Size before / after	# of nested attr.	Time (sec.)
`Balloons1`	5 / 16	42 / 64	80 / 22	0.455 / 0.152	3	1.08
`Balloons2`	5 / 16	42 / 64	80 / 22	0.455 / 0.150	3	1.07
`Balloons3`	5 / 16	40 / 64	80 / 42	0.455 / 0.260	3	1.14
`Balloons4`	5 / 16	42 / 64	80 / 22	0.455 / 0.149	3	1.07
`Hayes`	6 / 132	1 / 6	792 / 522	1.758 / 1.219	1	1.01
`Bupa`	7 / 345	0 / 7	2387 / 2387	7.234 / 7.234	0	4.40
`Balance`	5 / 625	56 / 65	3125 / 1120	6.265 / 2.259	4	21.48
`TA_Eval`	6 / 110	253 / 326	660 / 534	1.559 / 1.281	5	24.83
`Car`	7 / 1728	1870 / 1957	12096 / 779	51.867 / 3.157	6	469.47
`Flare`	13 / 365	11651 / 13345	4745 / 2834	9.533 / 5.715	4	6693.41

Table 7.10: Summary of NeT experimentations.

$nestAll_M(t)$ for any column X and all possible list of columns M of length m, where $m \geq l$.

Our preliminary results comparing the goodness of the *XSchema* obtained from NeT, and FT with that obtained from DB2XML v 1.3 [Tur99] appeared in [LMC01]. We further applied our NeT algorithm on several test sets drawn from UCI KDD[5] / ML[6] repositories, which contain a multitude of single-table relational schemas and data. Sample results are shown in Table 7.10. Two metrics are used as follows:

$$\begin{aligned} \text{NestRatio} &= \frac{\text{\# of successful nesting}}{\text{\# of total nesting}} \\ \text{ValueRatio} &= \frac{\text{\# of original data values}}{\text{\# of data values D in the nested table}} \end{aligned}$$

where D is the number of individual data values present in the table. For example, the D in the row $(\{1, 2, 3\}, a, 10)$ of a nested table is 5.

[5] http://kdd.ics.uci.edu/

[6] http://www.ics.uci.edu/~mlearn/MLRepository.html

Note that *NestRatio* denotes the efficiency of our optimization rules, while *ValueRatio* implies whether it was useful to perform nesting.

In our experimentation, we observed that most of the attempted nestings are succesful, and hence our optimization rules are quite efficient. In Table 7.10, we see that nesting was useful for all the data sets except for the `Bupa` data set. Also nesting was *especially* useful for the `Car` data set, where the size of the nested table is only 6% of the original data set. Time required for nesting is an important parameter, and it depends on the number of attempted nestings, and the number of tuples. The number of attempted nestings depend on the number of attributes, and increase drastically as the number of attributes increases. This is observed for the `Flare` data set, where we have to do nesting on 13 attributes. The NeT algorithm could nest only up to 4 attributes for feasibility reasons, it is actually possible to nest more for this data set.

7.7.2 CoT Results

For testing CoT, we need some well-designed relational schema where tables are interconnected via inclusion dependencies. For this purpose, we use the TPC-H schema v 1.3.0[7], which is an ad-hoc, decision support benchmark and has 8 tables and 8 inclusion dependencies. The IND-Graph for the TPC-H schema is shown in Figure 7.4.

CoT identifies two top-nodes - `part` and `region`. Then, CoT converted the TPC-H schema into the following *XSchema* $\mathbb{X}_{tpc} = (E, A, M, P, r, \Sigma)$. We show the definitions only for attributes of types `ID` and `IDREF`.

$$\begin{aligned} E &= \{part, partsupp, lineitem, orders, customer, supplier, nation, region\} \\ A(part) &= \{PartKey, Name, Mfgr, Brand, Type, Size, Container, \end{aligned}$$

[7] http://www.tpc.org/tpch/spec/h130.pdf

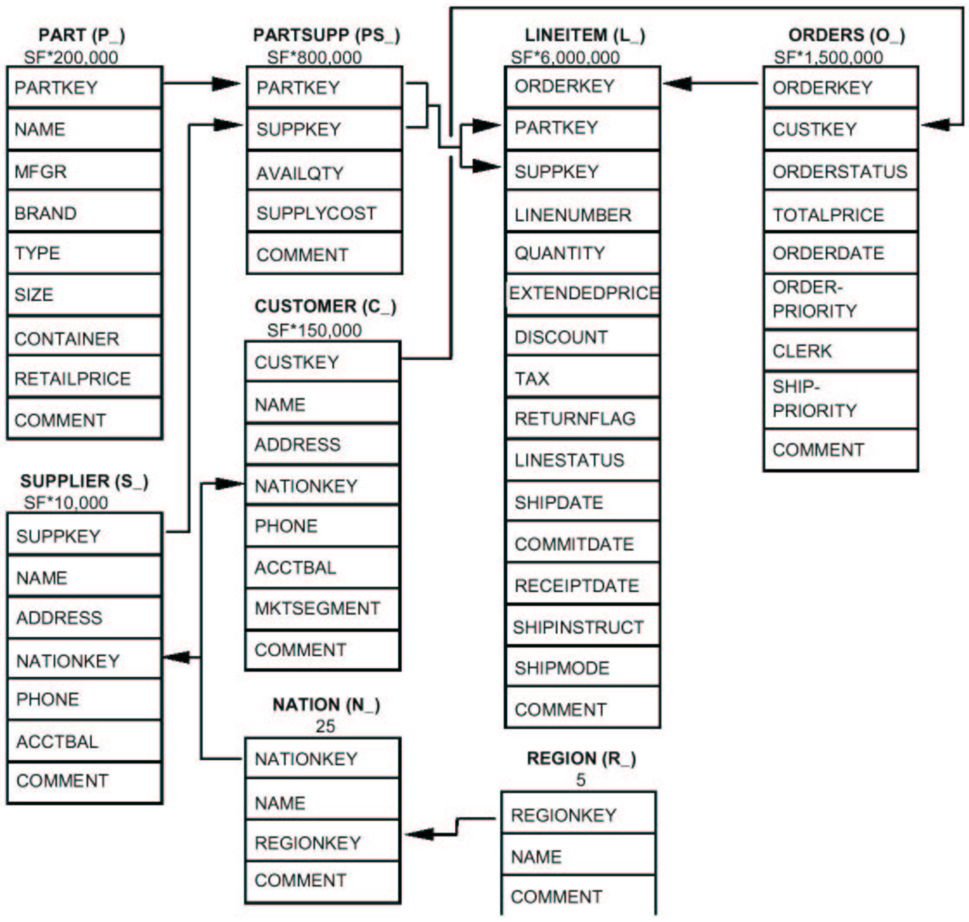


Figure 7.3: The TPC-H schema: the arrow → points in the direction of the 1-to-many relationship between tables.

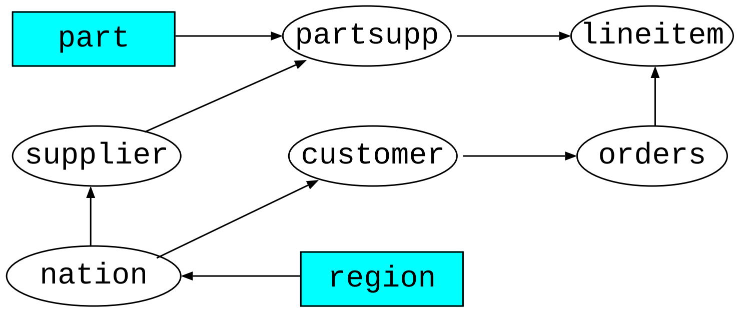


Figure 7.4: The IND-Graph representation of the TPC-H schema.

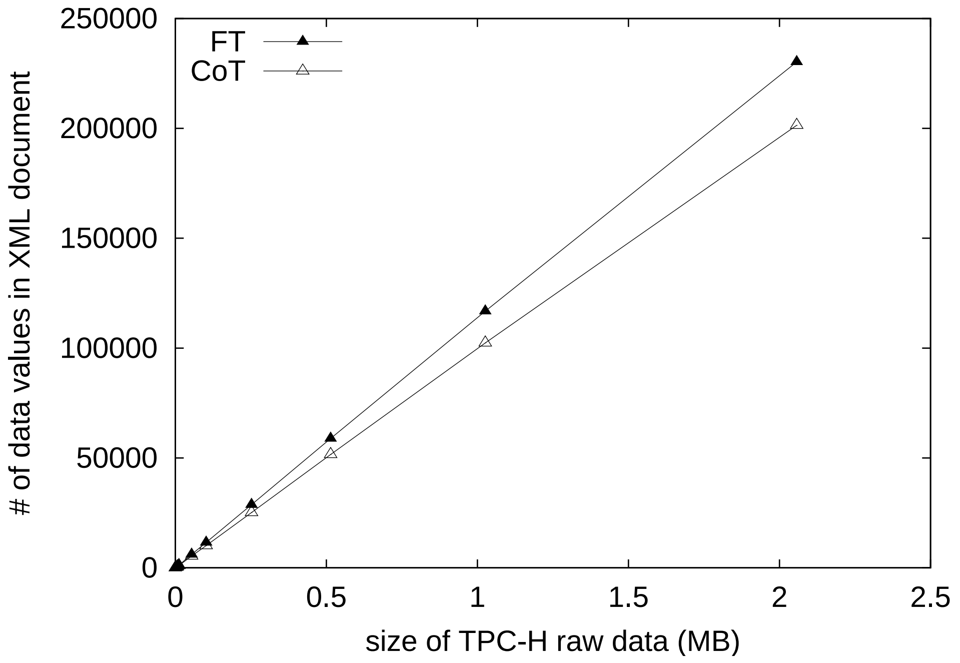


Figure 7.5: Comparison of XML documents generated by FT and CoT algorithms for TPC-H data.

$$
\begin{aligned}
& \quad RetailPrice, Comment\} \\
M(part) &= \epsilon \\
P(PartKey) &= (ID, \neg ?, \epsilon, \epsilon) \\
A(region) &= \{RegionKey, Name, Comment\} \\
M(region) &= \{nation^*\} \\
P(RegionKey) &= (ID, \neg ?, \epsilon, \epsilon) \\
A(nation) &= \{NationKey, Name, Comment\} \\
M(nation) &= \{supplier^*, customer^*\} \\
P(NationKey) &= (ID, \neg ?, \epsilon, \epsilon) \\
A(supplier) &= \{SuppKey, Name, Address, Phone, AcctBal, Comment\} \\
M(supplier) &= \{partsupp^*\}
\end{aligned}
$$

$$\begin{aligned}
P(SuppKey) &= (ID, \neg ?, \epsilon, \epsilon)\\
A(partsupp) &= \{Ref\_part, AvailQty, SupplyCost, Comment\}\\
M(partsupp) &= \{lineitem^*\}\\
P(Ref\_part) &= (IDREF, \neg ?, \epsilon, \epsilon)\\
A(lineitem) &= \{Ref\_orders, LineNumber, Quantity, ExtendedPrice,\\
&\quad Discount, Tax, ReturnFlag, LineStatus, ShipDate, CommitDate,\\
&\quad ReceiptDate, ShipInstruct, ShipMode, Comment\}\\
M(lineitem) &= \epsilon\\
P(Ref\_orders) &= (IDREF, \neg ?, \epsilon, \epsilon)\\
A(customer) &= \{CustKey, Name, Address, Phone, AcctBal, MktSegment, Comment\}\\
M(customer) &= \{orders^*\}\\
P(CustKey) &= (ID, \neg ?, \epsilon, \epsilon)\\
A(orders) &= \{OrderKey, OrderStatus, TotalPrice, OrderDate,\\
&\quad OrderPriority, Clerk, ShipPriority, Comment\}\\
M(orders) &= \epsilon\\
P(OrderKey) &= (ID, \neg ?, \epsilon, \epsilon)\\
r &= \{part, region\}
\end{aligned}$$

$$\begin{aligned}
\Sigma = \{&PartKey \overset{key}{\to} part, SuppKey \overset{key}{\to} supplier, \{Ref\_part, SuppKey\} \overset{key}{\to} partsupp,\\
&\{Ref\_orders, LineNumber\} \overset{key}{\to} lineitem, OrderKey \overset{key}{\to} orders,\\
&CustKey \overset{key}{\to} customer, NationKey \overset{key}{\to} nation, RegionKey \overset{key}{\to} region\}
\end{aligned}$$

Six of the eight inclusion dependencies are mapped using sub-element, and the remaining two are mapped using `IDREF` attributes. We believe that the *XSchema* produced by CoT is “more intuitive” than the relational schema we started with.

Figure 7.5 shows a comparison of the number of data values originally present in the database, and the number of data values in the XML document generated by FT and CoT. Because FT is a flat translation, the number of data values in the XML document generated by FT is the same as the number of data values in the original data. However, CoT is able to decrease the number of data values in the generated XML document by more than 12%.

Table 7.11 shows a portion of XML documents (generated by CoT) that conform to the *XSchema* $\mathbb{X}_{tpc}$. Note that CoT generated interwoven hierarchical structures in the final schema, instead of a flat schema.

7.8 Summary

We have presented two relational-to-XML conversion algorithms - NeT and CoT. The naive translation algorithm FT translates the "flat" relational model to "flat" XML model in a one-to-one manner. Thus FT does *not* use the non-flat features of the XML model, possible through regular expression operators such as "*" and "+". To remedy this problem, we first presented NeT, which uses the *nest* operator to generate a more precise and intuitive XML Schema from relational inputs. When poorly designed or legacy relational schema needs to be converted to XML format, NeT can suggest a fairly intuitive XML schema. However NeT is only applicable to a single table at a time, and *cannot* obtain a big picture of a relational schema where many tables are interconnected with each other. Our next algorithm CoT addresses this problem - CoT uses *semantic constraints* (especially inclusion dependencies) specified in the relational model to come up with a more intuitive XML Schema for the entire relational schema.

Thus our approaches have the following properties: (1) automatically infer a "good" XML Schema from a given relational schema, (2) remove redundancies that might be present in poorly designed or legacy relational schema (3) maintain

```
<TPC-H>
  <region RegionKey="1" Name="AMERICA" Comment="even, ironic theodolites">
    <nation NationKey="24" Name="UNITED STATES"
      Comment="blithely regular deposits serve furiously blithely regular!">
      <supplier SuppKey="19" Name="Supplier#000000019" Address="edZT3es,"
        Phone="34-278-310-2731" AcctBal="6150.38"
        Comment="quickly regular pinto beans mold blithely slyly pending.">
        <partsupp PartKey="2518" AvailQty="8018" SupplyCost="131.37"
          Comment="pending instructions sleep after the pending, regular.">
          <lineitem OrderKey="5999968" LineNumber="179124" Quantity="19"
            ExtendedPrice="259.28" Discount="0.06" Tax="0.00" ReturnFlag="N"
            ShipDate="1998-10-28" ReceiptDate="1998-11-25"
            ShipInstruct="COLLECT" ShipMode="SHIP" Comment="ironic"/>
        </partsupp>
      </supplier> ... <supplier/>
      <customer CustKey="117" Name="Customer#000000117" Address="uNhM,5Y"
        Phone="34-403-631-3505" AcctBal="3950.83" MktSegment="FURNITURE"
        Comment="ironic requests cajole furiously around the special.">
        <orders Ref_lineitem="5999968" OrderStatus="F"
          TotalPrice="354575.46" OrderDate="1992-12-24"
          OrderPriority="3-MEDIUM" Clerk="Clerk#000000736"
          ShipPriority="0" Comment="packages according to the regular"/>
      </customer> ... <customer/>
    </nation>
  </region>
  <part PartKey="1" Name="goldenrod spring peru powder" Mfgr="Manufacturer#1"
    Brand="Brand#13" Type="PROMO BURNISHED COPPER" Size="7"
    Container="JUMBO PKG" RetailPrice="901.00" Comment="final deposits s"
    Ref_partsupp="2518"/>
  ...
</TPC-H>
```

Table 7.11: TPC-H data (populated by DBGEN) published as an XML document that conforms to $\mathbb{X}_{tpc}$.

semantic constraints during translation. With a rapid adoption of XML standards among industries and majority of data still stored in relational databases, the need to correctly and effectively convert relational data into XML format is imminent. We believe our proposed methods are good additions to such a practical problem.

Our work in this Chapter concentrates on obtaining a “good” and “correct” XML schema. However there are still several other issues to be studied. Implementation issues (e.g., I/O cost, tagging strategy, nesting strategy) are very important. Early investigation on these issues is done in [Bou00]. Since our work in this Chapter proposes algorithms which can result in a fairly complex target XML schema as an output, studying an efficient implementation of our NeT and CoT algorithms is an important direction. Another direction of future research is studying the normalization theory of XML schema. By formally defining what is a “good” XML schema, one can devise better relational-to-XML conversion algorithms that result in normalized XML schema. Also our NeT algorithm performed only single attribute nesting. Multiple attribute nesting is another interesting research direction.

In publishing relational data as XML documents, implementation issues (e.g., I/O cost, tagging strategy, nesting strategy) are as important as an issue of inferring a right XML schema. Early investigation on the issue includes, for instance, works in [Bou00]. Since our work in this Chapter proposes algorithms which can result in a fairly complex target XML schema as an output, studying an efficient implementation of our NeT and CoT algorithms is an important direction. Another direction of future research is studying the normalization theory of XML schema. By formally defining what the “good” XML schema is first, one can devise better relational-to-XML conversion algorithms that result in normalized XML schema. the correctness of a converted schema between two models. For instance, using the notion of information capacity [MIR94], a theoretical analysis for the correctness of our translation procedures is possible; we can show that NeT and CoT algorithms are *equivalence preserving transformations*. However, we defer this detailed analysis to a later version. one can determine whether or not a relational schema A is *correctly* converted to an XML schema B. Given a variety of target XML schema language proposals, we believe that it is not always possible nor plausible to correctly convert A to B and the correctness aspect can be of help in a such setting.

CHAPTER 8

Conclusion

It has long been believed that exact matching in querying has limitations, especially for the intelligent query systems which try to mimic human discourse. In such a system, when the exact matching to the user query is not available, systems provide approximate matching instead. In both IR and DB communities, techniques to support such approximate matching have been the focus of active research in past. In this dissertation, we extend this idea to XML model so that systems can provide approximate "XML" answers when exact "XML" matches are not satisfactory.

The difficulties in doing so are that unlike relational models, in DB or a keyword list-based querying in IR settings, XML model adopts a hierarchical tree-shaped data model, and in turn XML queries composed by users are also naturally tree-shaped ones. Therefore, the problem of finding approximate answers that are close to the original queries in both structures and values becomes non-trivial. In this dissertation, we studied various issues related to this problem. Although we did not provide a complete ready-to-ship solution to the problem, we believe our understanding and findings presented in this dissertation will be a good foundation towards a generic solution in future.

In this dissertation, we first adopted the notion of query relaxation in relational model into XML model and discussed the types of possible XML query relaxation and their semantics. Second, we studied two fundamental issues that are related to support query relaxation using native XML engines. That is, we described some of the preliminary work on distance metric for trees and proposed a novel selectivity estimation method that takes the characteristics of query relaxation in XML model into consideration. Finally, we presented issues involved in converting data between XML and relational models and proposed three novel conversion algorithms that not only capture the original structures and values, but also well preserve the semantic constraints of the original schema. This is a necessary step to support query relaxation for XML model by way of using the mature relational database systems.

8.1 Future Work

Many interesting research directions are ahead.

- **Design and Implementation of Relaxation-enabled Query Language**: Once the full understanding of query relaxation for XML model has been established, it is important to provide a fine and declarative control of query relaxation to end users. For this purpose, one of the most common tools being used is the query language. Based on the relaxation framework in Chapter 3, we plan to build an XML query language that supports query relaxation. The major features of such language are the following:
  - The language is based on the standard XQuery [CFR01] query statements, and is downward compatible with the corresponding portion of XQuery FLWR statements.
  - The language allows the user to specify relaxation constructs (e.g. approximate values, conceptual terms, and preference list for certain

query condition) in a declarative manner.

  – The language allows the user to specify relaxation control constructs such as an unacceptable list for certain query condition, relaxation order for multiple relaxable conditions, minimum answer set size, etc. and allows the user to rank the XML answer sets based on the similarity metric specified in the query.

- **Better Method for Top-k XML Query**: In many applications, user specify some desirable target values without requiring the exact matches, but demanding the minimum number of answers to be ranked. In such a top-k query problem, one of the fundamental issue to solve is how to *relax* the original query in such s way that only one relaxation will guarantee the finding of k matches. In relational databases, techniques such as [CG99] have been available. However, no such technique has been developed for general XML model. The problem, of course, lies in the fact that the relaxation in XML model has to do with both values and structures of the query. A novel selectivity estimation technique proposed in Chapter 5 was aimed at serving the following control flow in query relaxation:

```
1. $N \leftarrow$ EstimateSelectivity($Q$)
2. While ($N < k$)
    (a) $Q \leftarrow$ Relax($Q$)
    (b) $N \leftarrow$ EstimateSelectivity($Q$)
```

  If a better technique, say **SingleRelax()**, towards the top-k query problem were available, the above pseudo-code could be rephrased as follows:

```
1. $N \leftarrow$ EstimateSelectivity($Q$)
```

```
2. If $(N < k)$
    (a) $Q \leftarrow$ **SingleRelax**$(Q)$
```

Note that the loop was removed due to the guarantee provided by the **SingleRelax()** method. Often, this removal implies significant gains in query relaxation.

References

[AAN01] A. Aboulnaga, A. R. Alameldeen, and J. F. Naughton. "Estimating the Selectivity of XML Path Expressions for Internet Scale Applications". In *VLDB*, Roma, Italy, Sep. 2001.

[ACS02] S. Amer-Yahia, S. Cho, and D. Srivastava. "Tree Pattern Relaxation". In *EDBT*, Prague, Czech Republic, Mar. 2002.

[AGM97] S. Abiteboul, R. Goldman, J. McHugh, V. Vassalos, and Y. Zhuge. "Views for Semistructured Data". In *Proc. of the Workshop on Management of Semistructured Data*, Tucson, Arizona, May 1997.

[AHD96] P. Arabie, L. J. Hubert, and G. De Soete (Eds). *"Clustering and Classification"*. World Scientific, 1996.

[AQM97] S. Abiteboul, D. Quass, J. McHugh, J. Widom, J.Wiener, and J. Widom. "The Lorel Query Language for Semistructured Data". *Int'l J. on Digital Libraries (IJDL)*, **1**(1):68–88, Apr. 1997.

[BC00] A. Bonifati and S. Ceri. "Comparative Analysis of Five XML Query Languages". *ACM SIGMOD Record*, **29**(1):68–77, Mar. 2000.

[BCD95] D. T. Barnard, G. Clarke, and N. Duncan. "Tree-to-tree Correction for Document Trees". Technical report, Queen's Univ. Computer Science Dept., Jan. 1995.

[BCN92] C. Batini, S. Ceri, and S. B. Navathe. *"Conceptual Database Design: An Entity-Relationship Approach"*. The Benjamin/Cummings Pub., 1992.

[BFR02] P. Bohannon, J. Freire, P. Roy, and J. Simeon. "From XML Schema to Relations: A Cost-Based Approach to XML Storage". In *IEEE ICDE*, San Jose, CA, Feb. 2002.

[BFW98] P. Buneman, W. Fan, and S. Weinstein. "Path Constraints in Semistructured and Structured Databases". In *ACM PODS*, Seattle, WA, Jun. 1998.

[BHP00] P. Bernstein, A. Halevy, and R. Pottinger. "A Vision of Management of Complex Models ". *ACM SIGMOD Record*, **29**(3):55–63, Dec. 2000.

[BKK00] S. Banerjee, V. Krishnamurthy, M. Krishnaprasad, and R. Murthy. "Oracle8i - The XML Enabled Data Management System.". In *IEEE ICDE*, San Diego, CA, Feb. 2000.

[BL01] A. Bonifati and D. Lee. "Technical Survey of XML Schema and Query Languages". Technical report, UCLA Computer Science Dept., Jun. 2001.

[BM01] P. V. Biron and A. Malhotra (Eds). "XML Schema Part 2: Datatypes". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-2/.

[Bou99] R. Bourret. "XML and Databases". Web page, Sep. 1999. http://www.rpbourret.com/xml/XMLAndDatabases.htm.

[Bou00] R. Bourret. "Data Transfer Strategies: Transferring Data between XML Documents and Relational Databases". Web page, 2000. http://www.rpbourret.com/xml/DataTransfer.htm.

[BPS00] T. Bray, J. Paoli, and C. M. Sperberg-McQueen (Eds). "Extensible Markup Language (XML) 1.0 (2nd Edition)". W3C Recommendation, Oct. 2000. http://www.w3.org/TR/2000/REC-xml-20001006.

[BR99] R. Baeza-Yates and B. Ribeiro-Neto. *"Modern Information Retrieval"*. Addison–Wesley, 1999.

[CAC94] V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. "From Structured Document to Novel Query Facilities". In *ACM SIGMOD*, Minneapolis, MN, Jun. 1994.

[CAM02] G. Cobena, S. Abiteboul, and A. Marian. "Detecting Changes in XML Documents". In *IEEE ICDE*, San Jose, CA, Feb. 2002.

[CCD99] S. Ceri, S. Comai, E. Damiani, P. Fraternali, S. Paraboschi, and L. Tanca. "XML-GL: a Graphical Language for Querying and Restructuring WWW Data". In *Int'l World Wide Web Conf. (WWW)*, Toronto, Canada, May 1999.

[CCH94] W. W. Chu, Q. Chen, and A. Huang. "Query Answering via Cooperative Data Inference". *J. Intelligent Information Systems (JIIS)*, **3**(1):57–87, Feb. 1994.

[CCH96] W. W. Chu, K. Chiang, C.-C. Hsu, and H. Yau. "Error-based Conceptual Clustering Method for Providing Approximate Query Answers". *Comm. ACM*, **39**(12):216–230, Dec. 1996. http://www.acm.org/pubs/cacm/extension.

[CD99] J. Clark and S. J. DeRose (Eds). "XML Path Language (XPath) Version 1.0". W3C Recommendation, Apr. 1999. http://www.w3.org/TR/xpath.

[CDF01] S. Comai, E. Damiani, and P. Fraternali. "Graphical Queries over XML Data". In *ACM Trans. on Information Systems (TOIS)*, (To appear), 2001.

[CFI00] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian. "XPERANTO: Publishing Object-Relational Data as XML". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.

[CFR01] D. Chamberlin, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu (Eds). "XQuery 1.0: An XML Query Language". W3C Working Draft, Jun. 2001. http://www.w3.org/TR/2001/WD-xquery-20010607/.

[CG99] S. Chaudhuri and L. Gravano. "Evaluating Top-k Selection Queries". In *VLDB*, Edinburgh, Scotland, Sep. 1999.

[Cha90] S. Chaudhuri. "Generalization and a Framework for Query Modification". In *IEEE ICDE*, Los Angeles, CA, Feb. 1990.

[CHC98] W. W. Chu, C-C. Hsu, A. F. Cardenas, and R. K. Taira. "Knowledge-Based Image Retrieval with Spatial and Temporal Construct". *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, **10**(6):872–888, 1998.

[CJK00] W. W. Chu, D. B. Johnson, and H. Kangarloo. "A Medical Digital Library to Support Scenario and User-Tailored Information Retrieval". *IEEE Trans. on Information Technology in Biomedicine*, **4**(2), Jun. 2000.

[CJK01] Z. Chen, H. V. Jagadish, F. Korn, N. Koudas, S. Muthukrishnan, R. T. Ng, and D. Srivastava. "Counting Twig Matches in a Tree". In *IEEE ICDE*, Heidelberg, Germany, Apr. 2001.

[Cla00] J. Clark (Eds). "XML Transformations (XSLT) Version 1.1". W3C Working Draft, Dec. 2000. http://www.w3.org/TR/xslt11.

[Cla01] J. Clark. "TREX – Tree Regular Expressions for XML". Web page, 2001. http://www.thaiopensource.com/trex/.

[CLC91] W. W. Chu, R. C. Lee, and Q. Chen. "Using Type Inference and Induced Rules to Provide Intensional Answers". In *IEEE ICDE*, Kobe, Japan, Apr. 1991.

[CM01] J. Clark and M. Murata (Eds). "RELAX NG Tutorial". OASIS Working Draft, Jun. 2001. http://www.oasis-open.org/committees/relax-ng/tutorial.html.

[CRF00] D. Chamberlin, J. Robie, and D. Florescu. "Quilt: An XML Query Language for Heterogeneous Data Sources". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.

[CRG96] S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. "Change Detection in Hierarchically Structured Information". In *ACM SIGMOD*, Montreal, Quebec, Canada, Jun. 1996.

[CTZ01] S-Y. Chien, V. J. Tsotras, and C. Zaniolo. "Efficient Management of Multiversion Documents by Object Referencing". In *VLDB*, Roma, Italy, Sep. 2001.

[CX00] J. M. Cheng and J. Xu. "XML and DB2". In *IEEE ICDE*, San Diego, CA, Feb. 2000.

[CYC94] W. W. Chu, H. Yang, K. Chiang, B. Ribeiro, and G. Chow. "CoGIS: A Cooperative Geographical Information System". In *SPIE Conf. on Knowledge-Based Artificial Intelligence Systems in Aerospace and Industry*, Orlando, FL, Apr. 1994.

[CYC96] W. W. Chu, H. Yang, K. Chiang, M. Minock, G. Chow, and C. Larson. "CoBase: A Scalable and Extensible Cooperative Information System". *J. Intelligent Information Systems (JIIS)*, **6**(2/3):223–259, May 1996.

[DFF99] A. Deutsch, M. F. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. "A Query Language for XML". In *Int'l World Wide Web Conf. (WWW)*, Toronto, Canada, May 1999.

[DFS98] A. Deutsch, M. F. Fernandez, and D. Suciu. "Storing Semistructured Data with STORED". In *ACM SIGMOD*, Philadephia, PA, Jun. 1998.

[EM01] D. W. Embley and W. Y. Mok. "Developing XML Documents with Guranteed "Good" Properties". In *Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, Nov. 2001.

[Fal01] D. C. Fallside (Eds). “XML Schema Part 0: Primer”. W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-0.

[Fel98] C. Fellbaum (Eds). *“WordNet: An Electronic Lexical Database”*. The MIT Press, 1998.

[FG01] N. Fuhr and K. Grossjohann. “XIRQL - An Extension of XQL for Information Retrieval”. In *ACM SIGIR*, New Orleans, LA, Sep. 2001.

[FK99] D. Florescu and D. Kossmann. “Storing and Querying XML Data Using an RDBMS”. *IEEE Data Eng. Bulletin*, **22**(3):27–34, Sep. 1999.

[FS00] W. Fan and J. Siméon. “Integrity Constraints for XML”. In *ACM PODS*, Dallas, TX, May 2000.

[FST85] P. C. Fischer, L. V. Saxton, S. J. Thomas, and D. V. Gucht. “Interactions between Dependencies and Nested Relational Structures”. *J. Computer and System Sciences (JCSS)*, **31**(3):343–354, Dec. 1985.

[FTS00] M. F. Fernandez, W.-C. Tan, and D. Suciu. “SilkRoute: Trading between Relations and XML”. In *Int’l World Wide Web Conf. (WWW)*, Amsterdam, Netherlands, May 2000.

[Gaa97] T. Gaasterland. “Cooperative Answering through Controlled Query Relaxation.”. *IEEE Expert*, **12**(5):48–59, 1997.

[Gal88] A. Gal. *“Cooperative Responses in Deductive Databases”*. PhD thesis, Dept. of Computer Science, Univ. of Maryland, College Park, 1988.

[GGM90] T. Gaasterland, P. Godfrey, and J. Minker. “Relaxation as a Platform for Cooperative Answering”. *J. Intelligent Information Systems (JIIS)*, **1**(3/4):293–392, 1990.

[GGR00] M. N. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. “XTRACT: A System for Extracting Document Type Descriptors from XML Documents”. In *ACM SIGMOD*, Dallas, TX, May 2000.

[GGW01] P. Ganesan, H. Garcia-Molina, and J. Widom. “Exploiting Hierarchical Domain Structure to Compute Similarity”. Technical report, Stanford Computer Science Dept. 2001-27, Jun. 2001. http://dbpubs.stanford.edu/pub/2001-27.

[GJK02] S. Guha, H. V. Jagadish, N. Koudas, D. Srivastava, and T. Yu. “Approximate XML Joins”. In *ACM SIGMOD*, Madison, WI, Jun. 2002.

[GKP94] R. L. Graham, D. E. Knuth, and O. Patashnik. *"Concrete Mathematics: A Foundation for Computer Science"*. Addison-Wesley Pub., 1994.

[God97] P. Godfrey. "Minimization in Cooperative Response to Failing Database Queries". *Int'l J. Cooperative Information Systems (IJCIS)*, **6**(2):95–149, Jun. 1997.

[Gro98] W3C XSL Working Group. "The Query Language Position Paper of the XSLT Working Group". In *WWW The Query Language Workshop (QL)*, Cambridge, MA, Dec. 1998.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *"Introduction to Automata Theory, Language, and Computation"*. Addison–Wesley, 2nd edition, 2001.

[HP00] H. Hosoya and B. C. Pierce. "XDuce: A Typed XML Processing Language". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.

[ISO86] ISO 8879. *"Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)"*, Oct. 1986.

[ISO00] ISO/IEC. *"Information Technology – Text and Office Systems – Regular Language Description for XML (RELAX) – Part 1: RELAX Core"*, 2000. DTR 22250-1.

[JK84] M. Jarke and J. Koch. "Query Optimization in Database Systems". *ACM Comp. Survey*, **16**(2):111–152, Jun. 1984.

[JKN99] H. V. Jagadish, O. Kapitskaia, R. T. Ng, and D. Srivastava. "Multidimensional substring selectivity estimation". In *VLDB*, Edinburgh, Scotland, Sep. 1999.

[JKS00] H. V. Jagadish, N. Koudas, and D. Srivastava. "On Effective Multi-Dimensional Indexing for Strings". In *ACM SIGMOD*, Dallas, TX, May 2000.

[JLS01] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson. "TAX: A Tree Algebra for XML". In *Int'l Workshop on Data Bases and Programming Languages (DBPL)*, Frascati, Rome, Sep. 2001.

[JNS99] H. V. Jagadish, R. T. Ng, and D. Srivastava. "Substring selectivity estimation". In *ACM PODS*, Philadelphia, PA, May-Jun. 1999.

[JS82] G. Jaeschke and H.-J. Schek. "Remarks on the Algebra of Non First Normal Form Relations". In *ACM PODS*, Los Angeles, CA, Mar. 1982.

[Kap82] S. J. Kaplan. "Cooperative Aspects of Database Interactions". *Artificial Intelligence*, **19**(2):165–187, Oct. 1982.

[KKR00] G. Kappel, E. Kapsammer, S. Rausch-Schott, and W. Retschitzegger. "X-Ray - Towards Integrating XML and Relational Database Systems.". In *Int'l Conf. on Conceptual Modeling (ER)*, pp. 339–353, Salt Lake City, UT, Oct. 2000.

[KM00] C.-C. Kanne and G. Moerkotte. "Efficient Storage of XML Data". In *IEEE ICDE*, San Diego, CA, Feb. 2000.

[KNS99] Y. Kanza, W. Nutt, and Y. Sagiv. "Queries with Incomplete Answers over Semistructured Data". In *ACM PODS*, Philadelphia, PA, May-Jun. 1999.

[KS01] Y. Kanza and Y. Sagiv. "Flexible Queries over Semistructured Data". In *ACM PODS*, Santa Barbara, CA, May 2001.

[KVI96] P. Krishnan, J. S. Vitter, and B. R. Iyer. "Estimating Alphanumeric Selectivity in the Presence of Wildcards". In *ACM SIGMOD*, Montreal, Quebec, Canada, Jun. 1996.

[LC00a] D. Lee and W. W. Chu. "Comparative Analysis of Six XML Schema Languages". *ACM SIGMOD Record*, **29**(3):76–87, Sep. 2000.

[LC00b] D. Lee and W. W. Chu. "Constraints-preserving Transformation from XML Document Type Definition to Relational Schema". In *Int'l Conf. on Conceptual Modeling (ER)*, pp. 323–338, Salt Lake City, UT, Oct. 2000.

[LC01] D. Lee and W. W. Chu. "CPI: Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema". *J. Data & Knowledge Engineering (DKE)*, **39**(1):3–25, Oct. 2001.

[Lin95] L. Lindbom. *"A Wiener Filtering Approach to the Design of Tracking Algorithms With Applications in Mobile Radio Communications"*. PhD thesis, Uppsala University, Nov. 1995. http://www.signal.uu.se/Publications/abstracts/a951.html.

[LMC01] D. Lee, M. Mani, F. Chiu, and W. W. Chu. "Nesting-based Relational-to-XML Schema Translation". In *Int'l Workshop on the Web and Databases (WebDB)*, Santa Barbara, CA, May 2001.

[LMM00] D. Lee, M. Mani, and M. Murata. "Reasoning about XML Schema Languages using Formal Language Theory". Technical report, IBM Almaden Research Center, RJ# 10197, Log# 95071, Nov. 2000. http://www.cs.ucla.edu/∼dongwon/paper/.

[LS02] D. Lee and D. Srivastava. "Counting Relaxed Twig Matches in a Tree". Technical report, UCLA Computer Science Dept., Feb. 2002.

[MIR94] R. J. Miller, Y. E. Ioannidis, and R. Ramakrishnan. "Schema Equivalence in Heterogeneous Systems: Bridging Theory and Practice (Extended Abstract)". In *EDBT*, Cambridge, UK, Mar. 1994.

[Mit97] T. M. Mitchell. *"Machine Leaning"*. McGraw-Hill Co., 1997.

[MLM01a] M. Mani, D. Lee, and R. D. Muntz. "Semantic Data Modeling using XML Schemas". In *Int'l Conf. on Conceptual Modeling (ER)*, Yokohama, Japan, Nov. 2001.

[MLM01b] M. Mani, D. Lee, and M. Murata. "Normal Forms for Regular Tree Grammars". Technical report, UCLA Computer Science Dept., 2001.

[MLM01c] M. Murata, D. Lee, and M. Mani. "Taxonomy of XML Schema Languages using Formal Language Theory". In *Extreme Markup Languages*, Montreal, Canada, Aug. 2001. http://www.cs.ucla.edu/∼dongwon/paper/.

[MM99] A. Malhotra and M. Maloney (Eds). "XML Schema Requirements". W3C Note, Feb. 1999. http://www.w3.org/TR/NOTE-xml-schema-req.

[Mot84] A. Motro. "Query Generalization: A Method for Interpreting Null Answers.". In *Expert Database Systems Workshop*, Kiawah Island, SC, Oct. 1984.

[Mot86] A. Motro. "SEAVE: A Mechanism for Verifying User Presuppositions in Query Systems". *ACM Trans. on Information Systems (TOIS)*, **4**(4):312–330, Oct. 1986.

[Mot90] A. Motro. "FLEX: A Tolerant and Cooperative User Interface to Databases". *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, **2**(2):231–246, Jun. 1990.

[MS01] S. Muench and M. Scardina (Eds). “XSLT Requirements Version 2.0”. W3C Working Draft, Feb. 2001. http://www.w3.org/TR/xslt20req.

[MSB98] M. Mitra, A. Singhal, and C. Buckley. “Improving Automatic Query Expansion”. In *ACM SIGIR*, Melbourne, Austrailia, Aug. 1998.

[Mur00a] M. Murata. “Hedge Automata: a Formal Model for XML Schemata”. Web page, 2000. http://www.xml.gr.jp/relax/hedge_nice.html.

[Mur00b] M. Murata. “RELAX (REgular LAnguage description for XML)”. Web page, Aug. 2000. http://www.xml.gr.jp/relax/.

[MW99] J. McHugh and J. Widom. “Query Optimization for XML”. In *VLDB*, Edinburgh, Scotland, Sep. 1999.

[NEF00] C. Nentwich, W. Emmerich, A. Finkelstein, and A. Zisman. “BOX: Browsing Objects in XML”. *Software Practice and Experience*, **30**(15):1661–1676, 2000. http://www.cs.ucl.ac.uk/staff/c.nentwich/Box/.

[NJ02] A. Nierman and H. V. Jagadish. “Evaluating Structural Similarity in XML Documents”. In *Int’l Workshop on the Web and Databases (WebDB)*, Madison,WI, Jun. 2002.

[OY87] Z. M. Özsoyoglu and L. Y. Yuan. “A New Normal Form for Nested Relations”. *ACM Trans. on Database Systems (TODS)*, **12**(1):111–136, Mar. 1987.

[PV99] Y. Papakonstantinou and P. Velikhov. “Enhancing Semistructured Data Mediators with Document Type Definitions”. In *IEEE ICDE*, pp. 136–145, Sydney, Austrialia, Mar. 1999.

[RHJ99] D. Raggett, A. L. Hors, and I. Jacobs (Eds). “HTML 4.01 Specification”. W3C Recommendation, Dec. 1999. http://www.w3.org/TR/html4/.

[RLS98] J. Robie, J. Lapp, and D. Schach. *“XML Query Language (XQL)”*. WWW The Query Language Workshop (QL), Cambridge, MA, Dec. 1998. http://www.w3.org/TandS/QL/QL98/pp/xql.html.

[Rob99] J. Robie (Eds). “XQL (XML Query Language)”. Web page, Aug. 1999. http://www.ibiblio.org/xql/xql-proposal.html.

[Sah00] A. Sahuguet. "Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.

[SLR98] D. Schach, J. Lapp, and J. Robie. "Querying and Transforming XML". In *WWW The Query Language Workshop (QL)*, Cambridge, MA, Dec. 1998.

[SSB00] J. Shanmugasundaram, E. J. Shekita, R. Barr, M. J. Carey, B. G. Lindsay, H. Pirahesh, and B. Reinwald. "Efficiently Publishing Relational Data as XML Documents". In *VLDB*, Cairo, Egypt, Sep. 2000.

[STH99] J. Shanmugasundaram, K. Tufte, G. He, C. Zhang, D. DeWitt, and J. Naughton. "Relational Databases for Querying XML Documents: Limitations and Opportunities". In *VLDB*, Edinburgh, Scotland, Sep. 1999.

[SYU99] T. Shimura, M. Yoshikawa, and S. Uemura. "Storage and Retrieval of XML Documents using Object-Relational Databases". In *Int'l Conf. on Database and Expert Systems Applications (DEXA)*, pp. 206–217, Florence, Italy, Aug. 1999.

[Tai79] K. Tai. "The Tree-to-Tree Correction Problem". *J. ACM*, **26**(3):422–433, 1979.

[TBM01] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn (Eds). "XML Schema Part 1: Structures". W3C Recommendation, May 2001. http://www.w3.org/TR/xmlschema-1/.

[Tur99] V. Turau. "Making Legacy Data Accessible for XML Applications". Web page, 1999. http://www.informatik.fh-wiesbaden.de/~turau/veroeff.html.

[TW00] A. Theobald and G. Weikum. "Adding Relevance to XML". In *Int'l Workshop on the Web and Databases (WebDB)*, Dallas, TX, May 2000.

[WDC01] Y. Wang, D. J. DeWitt, and J-Y. Cai. "X-Diff: A Fast Change Detection Algorithm for XML Documents". Technical report, U. Wisconsin, Computer Science Dept., 2001.

[WH60] B. Widrow and M. E. Hoff. "Adaptive Switching Circuits". *Institute of Radio Engineers, Western Electronic Show and Convention, Convention Record*, pp. 96–104, 1960.

[WLL01] X. Wu, T. W. Ling, M. L. Lee, and G. Dobbie. "Designing Semistructured Database Using ORA-SS Model". Unpublished Manuscript, 2001.

[Woo99] P. T. Wood. "Optimizing Web Queries Using Document Type Definitions". In *Int'l Workshop on Web Information and Data Management (WIDM)*, pp. 28–32, Kansas City, MO, Nov. 1999.

[ZC00] G. Zhang and W. W. Chu. "MDC: A Mixed-Type Data Clustering Algorithm". Technical report, UCLA Computer Science Dept., 2000.

[ZCF97] C. Zaniolo, S. Ceri, C. Faloutsos, R. T. Snodgrass, V.S. Subrahmanian, and R. Zicari. *"Advanced Database Systems"*. Morgan Kaufmann Pub., 1997.

[ZS89] K. Zhang and D. Shasha. "Simple Fast Algorithms for the Editing Distance Between Trees and Related Problems". *SIAM J. Comput.*, **18**(6):1245–1262, Dec. 1989.