USING TREE PATTERNS FOR FLEXIBLE HANDLING OF XML KEYS

By

Jing Wang

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
AT
MASSEY UNIVERSITY
PALMERSTON NORTH, NEW ZEALAND
DECEMBER 2007

Table of Contents

Abstract

Previous research shows that decision problem of XML keys is far more intricate than its relational counterpart. We review key constraints in the context of XML as introduced by Buneman et al [9, 10] and later refined by Hartmann et al [19, 20]. In this thesis we investigate three path expression languages for defining XML keys. They are path expression $\mathcal{PE}$, XPath expression $\mathcal{XE}$ and tree pattern $\mathcal{TP}$. We focus on a special tree pattern $\mathcal{TP}$ called *safe tree pattern*, denoted by $\mathcal{SP}$, which has been proved to be equivalent to XML keys where Q are in $\mathcal{XE}^{/,//}$,Q' are in $\mathcal{XP}^{/,//}$ and $P_1, \ldots, P_k$ are in $\mathcal{XE}^{/,*}$.

We propose a set of inference rules that is sound and complete for the implication of XML keys in the form of $\mathcal{SP}$. Our new containment result of $\mathcal{SP}$ is in PTIME. This result indicates a PTIME algorithm for deciding XML key implication, and shows that reasoning about XML key in $\mathcal{SP}$ is practically efficient.

Acknowledgements

I would like to express my gratitude to the many people who have made this thesis possible.

Most of all I would like to thank Professor Sven Hartmann, my supervisor, for his guidance, constant support and endless patience during this research.

Special thanks go to my colleges in the *(former)* Department of Information Systems at Massey University who have supported me throughout this year although it not possible to list all their names here individually.

Lastly, and most importantly, I am forever indebted to my family for their understanding and encouragement. I am grateful to my husband for his support. Without his this work would never have come into existence.

Jing Wang

December 1, 2007

Chapter 1

Introduction

1.1 Motivation

The World Wide Web Consortium promotes extensible Markup Language (XML) [8] and related standards, including XML Schema [14], XPath [5], XQuery [18], and XSLT [24]. XML is a very simple and flexible text format derived from SGML [13], which had been available for more than a decade without really influencing the development of Web applications. Originally, XML was designed to meet the challenges of large-scale electronic publishing, Nowadays, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere. XML is emerging as a data model in logical database design, also the XML documents are to serve double duty as databases. XML is self-explanatory and easily parsed. However, despite the high degree of syntactic flexibility, the formal semantic of the XML and related standards are not well developed. We need to define and prove theory that supports XML data modelling, data specification and data management. Various work have address the semantic of XML databases. Naturally

the study of integrity constraints has received increased attentions from the database research community because integrity constraints are an essential part of database scheme definition languages. They play fundamental roles of semantic specification, data validation, integrity control, query optimisation, and etc. Various work have attempted to define different classes of integrity constraints. Therefore, some classes of integrity constraints including keys have been made available for XML [9, 10, 20, 19].

Keys are of importance in databases. They are an integral part of database practice and theory in the context of the relational database model. In particular, keys are used to identify object within the database and reference one object from another. Keys and referential integrity constraints constitute an essential class of constraints on the validation of data. Since the same key constraints are required for XML databases, there is a need to investigate issues like XML key specifications and related issues such as satisfiability and implication problem. Intuitively, researchers attempt to apply the some approaches which have been used in the context of relation database model to XML data model. However, due to hierarchical structure of XML data, the problems of XML key definition and associated satisfiability and implication problems are much more complicated and harder than in the RDM. The *ID/IDREF* in XML DTD is too weak in term of expressive power. On the other hand side, the *key and keyref* in the XML Schema using full XPath expression is too complicate for reasoning about. There is a need to find a new way to define XML keys which are powerful and expressive enough and also still can be reasoned about efficiently. This is the motivation of our work.

Various approaches for XML keys have been proposed, e,g, [9, 10, 20, 19]. Independently from any specifications such a document type definition (DTD) or XML

Schema definition (XSD), these keys are based on the representation of XML data as trees and have been defined in term of path expressions. Their associated decision problems such as satisfiability and implication are of our research interests.

1.2 Objectives

The main goal of this thesis is to investigate key implication problem by using XML tree pattern. In this thesis we continue the study of path-expression-based key specification presented in [9, 10, 20, 19]. We review XML key constraints and observe that in order to define XML key constraints the choice of a path language can be crucial. Generally speaking, We need a suitable path language to select nodes in XML which is expressive enough to allow reasonable navigation and simple enough to allow efficient reasoning.

- The first part of our work is to examine different approaches of expressing node selection queries, study translation and discover equivalences between them.

- In the second part of our work, we focus on tree patterns as a convenient way to express paths, thus, to define XML keys. Our focus is on defining a *safe tree pattern* and reasoning about the containment problem and eventually investigating its computational complexity.

- Last, in accordance with other work relating to XML keys, we are interested in a generalised XML key in the form of the *safe tree pattern*. We provide a set of inference rules for the implication of XML keys in the *safe tree pattern*. We need to prove that their (finite) implication problem is finitely axiomatisable. Consequently we can then apply the result of the containment problem of *safe*

tree pattern to the key implication problem. The time complexity of the *safe tree pattern* containment is in PTIME, hence, the implication problem can be decided in PTIME as well. Therefore, we show that XML keys defined in the form the *safe tree pattern* can be maintained efficiently by database systems for XML applications.

1.3 Outline of the Thesis

The reminder of this thesis is organised as follows: In Chapter 2 we present some preliminary definitions used throughout this thesis, illustrate the lack of effective and efficient integrity constraints in the current XML scheme specification languages, namely, DTD and XSD and at the end of this chapter we present a running example used in this thesis. In the next two chapters three path languages, namely, Path expression $\mathcal{PE}$, XPath expression $\mathcal{XE}$ and tree pattern $\mathcal{TP}$, have been examined for selecting nodes in XML trees. We limit our study of query languages to *downward* queries only. Upward paths, for example, *parent*, *ancestor* or *sibling* paths, are not relevant to our study and we leave them out. We prove semantic equivalence between the query languages and show how to translate from one to another. The query containment problem is our main interest of study. Chapter 5 we present our result of using *safe* tree pattern handling XML keys. In Chapter 6 we conclude the work. Related work is discussed in Chapter 6.1 and finally we outline some possible research directions in Chapter 6.3

Chapter 2

Preliminaries

2.1 XML Tree Model

In this section we recall the basics of the XML tree model, node addressing and the notion of value equality, which is essential for defining keys in the context of XML databases.

XML data is commonly modelled by trees with node labels from an infinite set Λ of symbols. Consider a tree T with node set V_T, edge set E_T, and a distinguished root node $r_T \in V_T$. The *size* $|T|$ of T is the cardinality of its node set. T is *finite* if V_T is finite, and *empty* if V_T consists of the root node only. All trees considered in this thesis are finite unless stated otherwise.

By E_T^+ we denote the transitive closure, and by E_T^* the reflexive and transitive closure of E_T. Recall that for each node $v \in V_T$ there is a unique (directed) path of edges from the root r_T to v, hence r_T is an ancestor of all nodes in T.

An *XML tree* is a tree T with a mapping $lab_T : V_T \to \Lambda$ that assigns nodes their labels. The symbols in Λ represent element labels $\mathbf{E}$, attribute labels $\mathbf{A}$, and text

labels **S**. We also assume that these sets are pairwise disjoint, hence, $E \cap A \cap S = \emptyset$.

An *XML data tree* is an XML tree T with a partial mapping $val_T : V_T \rightarrow \Lambda$ that assigns string values to each attribute and text node. The strings in Λ represent attribute and text values. We use L_T to denote the domain of val_T and call its members the *data hosts*.

2.1.1 Nodes in XML trees

The idea to regard XML data as trees goes back to the Document Object Model (DOM) [2]. XML data is commonly stored and exchanged in XML documents. DOM describes an XML document as a hierarchical structure of nodes. There are three important kinds of nodes which we are interested in: element nodes (**E**), attribute nodes (**A**), and text nodes (**S**). Element nodes have a label and may have children:

- a text node (PCDATA)
- a set $A \subseteq \Lambda$ of attributes notes
- a list $E \subseteq \Lambda$ of element nodes
- each child w induces a edge (v, w) in the XML tree

Attribute nodes and text nodes are terminal, that is, they are leaves. Each attribute or text node v has a label, $lab_T \in \Lambda$ and carry a value, $val_T \in S$. A consequence of the DOM model is that form the root r_T to a node v there is a unique path, which allows us to identify and locate a node in an XML tree.

2.1.2 Value Equality

In order to define key constraints there is a need to separate notations of node equality and value equality in term of XML tree model. We now define value equality for pairs of nodes in XML trees. Informally, two nodes, u and v, are value-equal if they have the same label, the same string value (if defined), and there is a bijection between the children of u and the children of v that associates value-equal nodes. More formally, we have the value equality definition as follows:

Definition 2.1.1. (Value Equality) $u, v \in V_T$ are *value-equal*, denoted by $u =_v v$, if the subtrees of T rooted at u and at v are isomorphic by an isomorphism that is the identity on Λ.

For two sets U, V of nodes we will also consider the *value-intersection* $U \cap_v V$ consisting of all node pairs (u, v) with $u \in U$, $v \in V$ and $u =_v v$. It is written as follows: $U \cap_v V = \{(u, v) \mid u \in U, v \in V, u =_v v\}$.

2.2 Overview of XML

In this section, we present an overview of XML, emphasising the importance of having efficient integrity constraints definitions for XML.

Internet applications and the new way in which they handle data have had a significant impact on database theories and practice. Over last ten years the eXtensible Markup Language (XML) has gradually gained its dominant importance as a universal standard for data representation and exchange on the Internet. Challenges and research directions which the XML research community currently faces have been outlined and discussed in [12, 31, 34, 35].

In [12] Ceri discussed the challenges from the following three perspectives:

- **XML as a data representation standard** We can simply divide currently available XML documents into two categories: well-formed and valid. It is ongoing debate among database researchers on the representation of XML data: should it be "falt" and schema independent, or should it be more complex and influenced by schema? It is obvious that the answer to the latter one is "Yes". In the context of the relational data model data definition and manipulation languages have been studied thoroughly. The same solid semantic foundation is desirable for the XML databases, which requires abstractions for modelling, constraining, querying, and updating. In Section 2.3 two dominant schema languages are discussed.

- **XML as a data exchange standard** "Data on the Web" proposed tremendous challenges of data exchange and integration. However this topic is beyond the scope of this thesis.

- **XML as a basis for building a new repository technology** In the context of relational databases, a DBMS provides retrieving, updating as well as transactions, concurrency, distribution and recovery, in a controlled environment. However, "data on the web" is free-evolving, ever-changing collection of data sources in various forms and formats. Tackling these issues in the XML context will improve repository technology for XML databases. Same as above we are not going to explore this perspective in details in the thesis.

It is important to understand that these three perspectives are closely related to each other although we focus on XML data specifications only. For example, an

expressive and efficient XML key specification would provide theoretical foundation for improving performance and maintenance of XML databases in practice.

2.3 Schema Languages for XML

Any data-central system must provide a data definition language as well as a data manipulation language. In the context of relational database management systems these languages have been well studied. However, the flexibility of the XML data comes at a price: the loss of schema. Data specifications for XML usually comes in the forms of a type specification, e.g., a Document Type Definition (DTD) [8], part of the original W3C recommendation defining XML, or an XML Schema Definition (XSD) [14], W3C recommendation which supersedes DTD. Additionally, there are a number of schema languages (type systems) for XML, including: RELAX NG, which is rooted on finite tree automata (FTA), and DataGuide, closely related to the Lore project at Stanford University. For the purpose of this thesis we only discuss the first two. A comprehensive discussion on schemas for XML can be found in [34].

2.3.1 Document Type Definition (DTD)

Essentially, a DTD is an extended context-free grammar. In [3], the definition of DTD has been formalised as follows: A DTD is a tuple $D = (E, A, P, R, r)$, where:

- E is a finite set of *element types*
- A is a finite set of *attribites*. A and E are disjoint.
- For each $\tau \in E$, $P(\tau)$ is a regular expression α, called the *element type definition* of τ: $\alpha ::= S \mid \tau' \mid \epsilon \mid \alpha|\alpha \mid \alpha,\alpha \mid \alpha^*$ where S denotes the *string*

type, $\tau' \in E$, ϵ is the empty element, and "|", ",", and "*" denote union, concatenation, and the Kleene closure;

- For each $\tau \in E$, $R(\tau)$ is a set of attributes in A;
- $r \in E$ and is called the element type of the root.

Normally the element type is denoted by τ, and attribute by $@l$. Let $D = (E, A, P, R, r)$ be a DTD, an XML tree T confirming to D, written $T \models D$.

A key containing exactly one field is called unary. In DTDs, unary key (ID) and foreign key (IDREF) constraints are defined as ID/IDREF attributes. However, the restrictions of this approach are significant: only one ID attribute per element type; the value of the ID must be unique in the entire XML document, no space is allowed in the value of ID attribute.

2.3.2 XML Schema

XML Schema defines both a type system and a class of integrity constraints. In contrast to simple type system (PCDATA), XML schema support more sophisticated type system: atomic types such as string, integer, and date etc., complex type constructors, (e.g., sequence, choice) and inheritance mechanisms (e.g., extension, restriction). The constraints defined in XML schema using XPath including key, foreign key and unique constraints improve the ID/IDREF mechanism in DTDs. First, the scope of a key does not need be the entire XML document. Second, a key may have several key fields.

2.3.3 Comparison between DTD and XML Schema

A schema language needs to satisfy three criteria to be useful.

1. First, it must provide sufficient *expressiveness* such that most syntactic requirements can be formalised in the language.

2. Second, it must be possible to implement *efficient* schema processors. The time and space requirements for checking validity of a document should ideally be linear in the size of the document.

3. Third, the language must be *comprehensible*. It should be easily adopted by the database community.

Using these three criteria now we compare DTD and XML Schema. The third criteria is trivial for our case and we omit it. Regarding the expressiveness it is obvious that XML Schema is more powerful language compared with DTD. Regarding the efficiency, we are interested in the consistency, or satisfiability problem for a given specification. If we only consider the schema itself, the problem is decidable, see Table 2.1 below. However it becomes complicated when we take constraints, e.g., key constraints, into consideration.

2.4 Constraints

Constraints are essential ingredients to databases. Their applications are schema design, data validation, query optimisation, and choice of efficient data storage and access methods. The most common database constraints are *functional dependencies* and *inclusion constraints* [34] in the relational datebase context. In this thesis

we study inclusion constraints, more precisely key constraints and its implication problems. Some classes of constraints have been defined [16, 17].

In XML, key constraints can be expressed using path expressions. They involve the data values associated to the leaves of XML documents. Vianu [34] pointed out that key constraints can be formlised as a pair $(q, \{p_1, ..., p_n\})$ where q and the p_i's are path expressions. Intuitively, q identified the element e to which the key constraint applies and $p_1, ..., P_n$ the nodes whose data values collectively identify each elements. This idea have been refined and formlised in Buneman's work [9, 10]. Naturally, after having key definition, we need to study decision problems for XML keys. The implication problem for key constraints is harder than in the relational case simply because it involves reasoning about path expression, i.e., path containment problem. In some work like [27] show that restriction on the path expressions could lead to a PTIME algorithm. However, the general case is coNP-complete [?, 27].

Now we take both of them into consideration. That is, whether a given set of constraints and a schema (DTD or XML Schema) are satisfiable by an XML document. Formally, the *satisfiability problem* is defined as follows:

Definition 2.4.1. Σ denotes a set of constraints, D denotes a definition or specification, the satisfiability problem is to determine if there is an XML tree T such that $T \models \Sigma$. and $T \models D$.

In [3], the authors demonstrated that the semantics of XML Schema constraints makes the satisfiability problem is very costly. Reasoning key constraints in the presence of DTD or XSD is intractable. The main results are summarised in Table 2.1

In general, the satisfiability problem is undecidable for a given set of constraints

	DTD	XML Schema
Keys and foreign keys	undecidable	undecidable
Unary keys and foreign keys	NP-complete	PSPACE-hard
Key only	linear time	NP-hard
No constraints	linear time	linear time

Table 2.1: Complexity of the satisfiability problem.

and a specification; for extremely restricted DTDs and constraints, it is still rather expensive (NP-hard and PSPACE-hard). In particular, with only unary keys, the satisfiability problem is NP-hard under the XML schema semantics, in contract to its linear-time decidability under the DTD.

2.5 An Example

As an example let us consider an on-line ticketing agency system named "*WhereToGo*". WhereToGo sells tickets for a variety of productions like concerts, movies, musicals, and theatre plays. Typically each production is shown at various locations and times all over New Zealand. We try to capture the following requirements in our DTD and XSD:

- A production has a title and a description. The title is unique. A production may optionally have an official Web site and/or an official trailer provided as URLs.

- There are different kinds of productions including concerts, movies, musicals, and theatre plays.

- For particular kinds of productions there is additional information available. A movie, for example, has a one or more directors and a cast listing all the actors.

- For each production there are one or more sessions scheduled. Each session has a venue, a date and a starting time.
- Each venue (such as a cinema or a theatre) is located in one and only one city. A city can have several venues. Each venue has a name, an address, a contact phone number and, optionally, an official website.

It should be noted that here we do not concern ourselves with the issue of whether some information are better modelled as an element or an attribute.

Example 2.5.1. *The DTD for this scenario is shown in Fig 2.1 and the XML tree model for the DTD is shown in Fig 2.2. We model* Production *and* Venue *as elements. we could see for each venue there is a unique VenueID defined as attribute type ID, and then in the Session element we have a VenueID as IDREF, which is used for references to a VenueID attribute in element Venue. In this case, ID and IDREF provide only very primitive support of key and key reference.*

Example 2.5.2. *An XML database instance conforming to the DTD above is defined, see Fig 2.3.*

Example 2.5.3. *A graphic representation of the corresponding XML schema definition (XSD) is shown in Fig 2.4, which is generated by XMLSpy from Altova. Due the limitation of this representation attributes are not shown. But we can see that* occurrences*of elements are presented in XML Schema. For example, solid lines represent mandatory elements, exactly one and at least one; dotted lines denote optional elements, zero and at most one. Additional, occurrence of an element is shown below the element rectangle, e.g.,* 0..5 *under the element Actor tells us that for each movie or play there is no actor or maximum five actors information stored.*

```
<?xml version="1.0" encoding="UTF-8"?>

<!--the root element is named "DB"-->
<!ELEMENT DB (Production*, Venue*)>

<!--this part is about production declaration-->
<!ELEMENT Production (Description, Website?,
Trailer?,
                (Movie|Concert|Play)?, Session+)>
<!ATTLIST Production
        Title CDATA #REQUIRED>

<!ELEMENT Description (#PCDATA)>
<!ELEMENT Website (#PCDATA)>
<!ELEMENT Trailer (#PCDATA)>

<!ELEMENT Movie (Director+,Actor*)>
<!ATTLIST Movie
        Year CDATA #IMPLIED
        Minutes CDATA #IMPLIED
        Rating CDATA #IMPLIED>
<!ELEMENT Director (#PCDATA)>
<!ELEMENT Actor (#PCDATA)>
<!ELEMENT Concert (Artist+, Song*)>
<!ELEMENT Artist (#PCDATA)>
<!ELEMENT Song (#PCDATA)>
<!ELEMENT Play (Writer+, Director+, Actor*)>
<!ELEMENT Writer (#PCDATA)>

<!ELEMENT Session EMPTY>
<!ATTLIST Session
        VenueID IDREF #REQUIRED
        Date CDATA #REQUIRED
        StartTime CDATA #REQUIRED>

<!--this part is about venue declaration-->
<!ELEMENT Venue (Website?)>
<!ATTLIST Venue
        VenueID ID #REQUIRED
        Name CDATA #REQUIRED
        City CDATA #REQUIRED
        Address CDATA #REQUIRED
        Phone CDATA #IMPLIED>
```

Figure 2.1: *Example: DTD for WhereToGo ticketing agency*

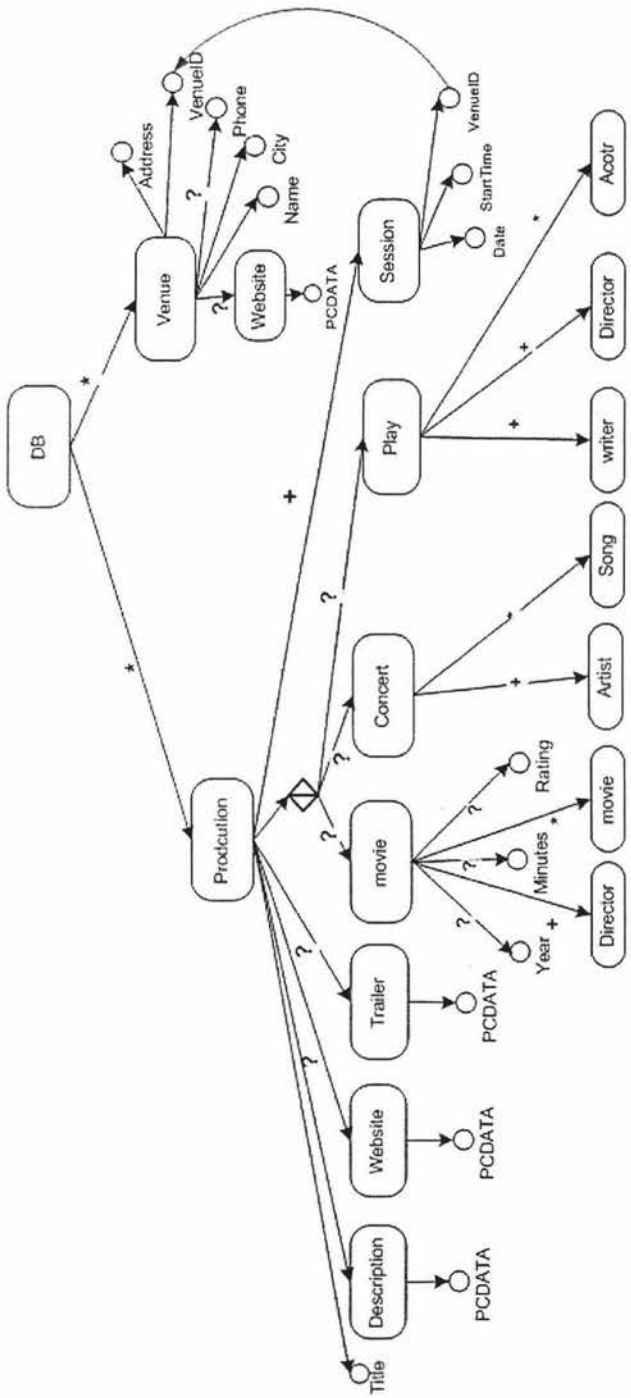

Figure 2.2: *Example: Scheme Tree for WhereToGo ticketing agency*

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE DB SYSTEM "wheretogo.dtd">
<DB>
        <Production Title="Madagascar">
                <Description>          ... </Description>
                <Movie Year="2005" Minutes="86" Rating="PG">
                        <Director>Eric Darnell</Director>
                        <Director>Tom McGrath</Director>
                </Movie>
                <Session VenueID="pn_downtown" Date="2006-04-01" StartTime="2pm"/>
                <Session VenueID="pn_downtown" Date="2006-04-02" StartTime="2pm"/>
                <Session VenueID="wn_state" Date="2006-04-02" StartTime="2pm"/>
        </Production>

        <Production Title="Pooh's Heffalump Movie">
                <Description>...</Description>
                <Website>http://www.poohsheffalump.com/</Website>
                <Trailer>http://trailer.poohsheffalump.com/</Trailer>
                <Movie Year="2005" Minutes="67" Rating="G">
                        <Director>Frank Nissen</Director>
                </Movie>
                <Session VenueID="pn_downtown" Date="2006-04-01" StartTime="10am"/>
                <Session VenueID="wn_suter" Date="2006-04-02" StartTime="11am"/>
        </Production>

        <Production Title="The Pink Panther">
                <Description>...</Description>
                <Website>http://www.pinkpantherthemovie.com/</Website>
                <Trailer>http://www.apple.com/trailers/mgm/the_pink_panther/</Trailer>
                <Movie Year="2006" Minutes="105" Rating="PG">
                        <Director></Director>
                        <Actor>Steve Martin</Actor>
                        <Actor>Jean Reno</Actor>
                        <Actor>Beyoncé Knowles</Actor>
                        <Actor>Kevin Kline</Actor>
                        <Actor>Kristin Chenoweth</Actor>
                </Movie>
                <Session VenueID="pn_downtown" Date="2006-04-01" StartTime="8pm"/>
                <Session VenueID="pn_downtown" Date="2006-04-02" StartTime="8pm"/>
                <Session VenueID="wn_state" Date="2006-04-01" StartTime="8pm"/>
                <Session VenueID="wn_state" Date="2006-04-02" StartTime="8pm"/>
        </Production>

        <Production Title="Patangi Loi">
                <Description>...</Description>
                <Play>
                        <Writer>Sebastian Hurrell</Writer>
                        <Director>Simon Ferry</Director>
                </Play>
                <Session VenueID="pn_centre" Date="2006-04-01" StartTime="7pm"/>
                <Session VenueID="pn_centre" Date="2006-04-02" StartTime="7pm"/>
        </Production>

        <Venue VenueID="pn_downtown"
                Name="Downtown Cinema" City="Palmerston North" Address="Broadway" Phone="(06) 355 5555">
                <Website>http://www.dtcinemas.co.nz</Website>
        </Venue>

        <Venue VenueID="pn_centre"
                Name="Centrepoint Theatre" City="Palmerston North" Address="Church St" Phone="(06) ">
                <Website>http://www.centrepoint.co.nz</Website>
        </Venue>

        <Venue VenueID="wn_state"
                Name="State Cinema Centre" City="Wellington" Address="Trafalgar St" Phone="(04) 5445726"/>

        <Venue VenueID="wn_suter"
                Name="Suter Cinema" City="Wellington" Address="Queens Gardens Bridge St" Phone="(04) 5413333">
                <Website>http://www.suter.co.nz</Website>
        </Venue>
</DB>
```

Figure 2.3: *Example: XML database for WhereToGo ticketing agency*

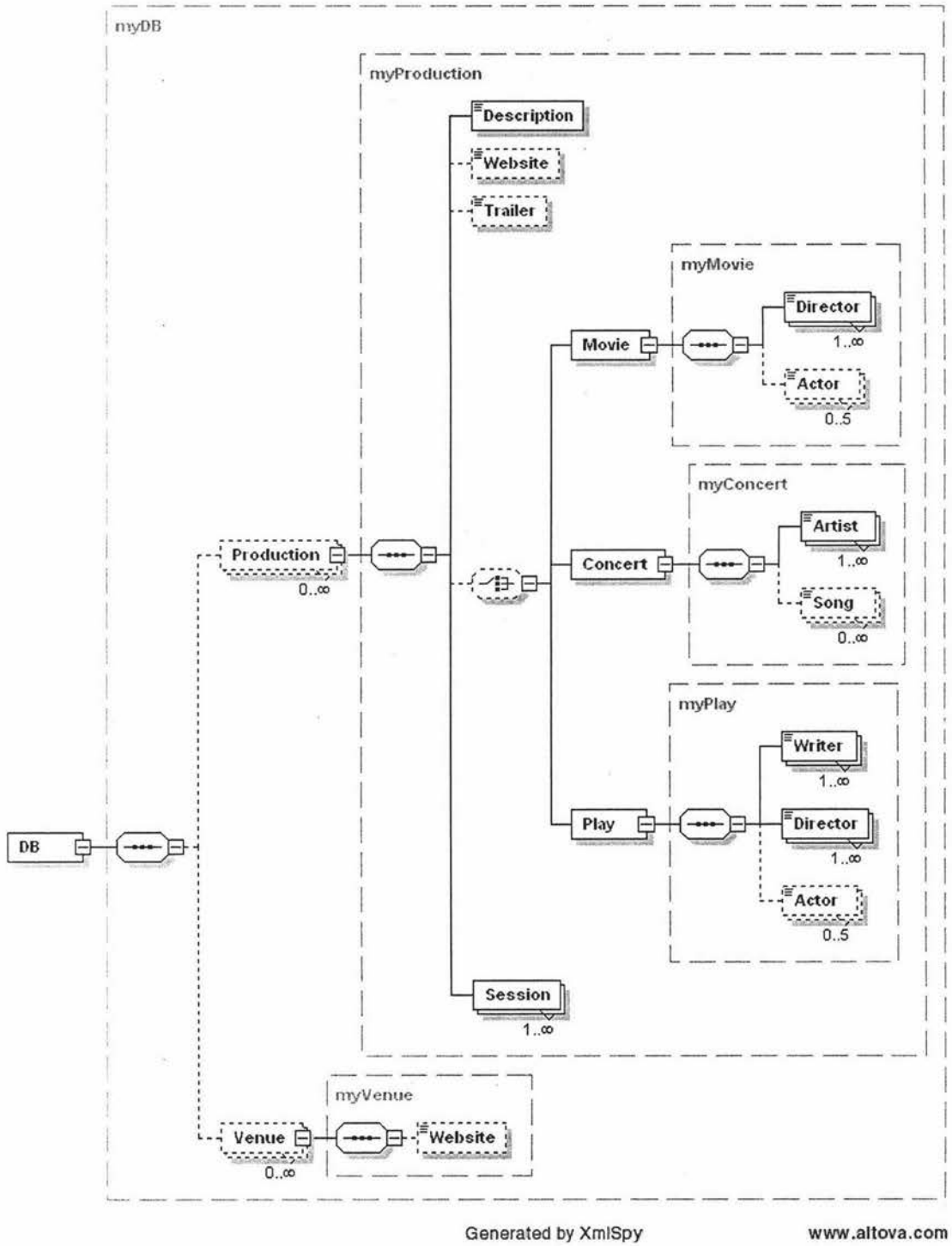

Figure 2.4: *Example: XML Scheme for WhereToGo ticketing agency*

```
<xs:key name="VenueID">
        <xs:selector xpath="DB/Venue"/>
        <xs:field xpath="@VenueID"/>
</xs:key>
<xs:keyref refer="VenueID" name="relatedVenueID">
        <xs:selector xpath="DB/Production/Session"/>
        <xs:field xpath="@VenueID"/>
</xs:keyref>
...
```

Figure 2.5: *Example: key and keyref defintion in XSD*

Example 2.5.4. *Fig 2.5 shows the key and keyref definition in XML Schema.*

Chapter 3

Node Selection Queries

3.1 Selecting Nodes in XML Trees

For a tree T, a *node selection query* Q defines a mapping $[\![Q]\!]_T : V_T \to 2^{V_T}$ that assigns every node $v \in V_T$ a subset of V_T, called the *selected nodes*, that may be seen as the result of executing the query at node v.

It remains to find a convenient way to express node selection queries. Depending on the respective application one wants to support queries that are expressive enough to be practical, yet sufficiently simple to be reasoned about efficiently. In this work we will discuss three approaches and study translations between them.

3.2 Query Containment

A problem that has been widely studied due to its immediate practical relevance is the containment problem. Two node selection queries Q, Q' are *contained*, denoted by $Q \sqsubseteq Q'$, if for every data tree T and every node $v \in V_T$ we have that $[\![Q]\!]_T(v)$ is a

subset of $[\![Q']\!]_T(v)$. Two queries are *equivalent*, denoted by $Q \equiv Q'$, if they contain one another. The *containment problem* for a class of queries asks to decide containment, while the *equivalence problem* asks to decide equivalence for the query class under inspection.

We call a node selection query Q *downward* if for any data tree T and any node $v \in V_T$ each selected node is a descendant-or-self of v. It should be noted that all node selection queries discussed in the sequel are downward. For them we record the following observation.

Proposition 3.2.1. *Two downward node selection queries Q, Q' are contained if and only if for every data tree T we have that $[\![Q]\!]_T(r_T)$ is a subset of $[\![Q']\!]_T(r_T)$.*

3.3 Path Expressions

Path expressions are a popular way of describing node selection queries. Path expressions are finite sequences of symbols. Table 3.1 defines the path language $\mathcal{PE} = \mathcal{PE}^{\{\text{-},\text{-}^*,\text{.}\}}$. Herein, ℓ denotes any symbol from Λ, ε the empty path expression, "." the concatenation of two path expressions, "_" the single symbol wildcard, and "_*" the variable length wildcard.

Path Language	Syntax
$\mathcal{PE} = \mathcal{PE}^{\text{-},\text{-}^*,\text{.}}$	$Q \to \ell \mid \varepsilon \mid \_^* \mid \_ \mid Q.Q$

Table 3.1: The path language $\mathcal{PE}$.

The semantics of $\mathcal{PE}$ expressions as node selection queries is defined inductively

as follows:

$$[\![\ell]\!]_T(v) := \{w \mid (v,w) \in E_T, lab_T(w) = \ell\}$$

$$[\![\_]\!]_T(v) := \{w \mid (v,w) \in E_T\}$$

$$[\![\_^*]\!]_T(v) := \{w \mid (v,w) \in E_T^*\}$$

$$[\![\varepsilon]\!]_T(v) := \{v\}$$

$$[\![Q_1.Q_2]\!]_T(v) := \{x \mid w \in [\![Q_1]\!]_T(v), x \in [\![Q_2]\!]_T(w)\}$$

When studying XML keys, Buneman et al. [9, 10] used the path languages $\mathcal{PL}_s = \mathcal{PE}^{\cdot}$ and $\mathcal{PL} = \mathcal{PE}^{\cdot,\_^*}$ that are given in Table 3.1. That is, "_" could not be used directly, but only in combination with the Kleene star "*".

Example 3.3.1. *Recall the XML tree in Fig 2.2 in Chapter 2.5 we can have the following path expressions:*

1. *DB.Production.Movie.Actor*
2. *DB.Production._.Actor*
3. *_*.Actor*

3.4 XPath Queries

XPath is the industry standard for expressing node selection queries in XML trees and XPath queries form an essential element of more sophisticated query languages like XQuery and XSLT. Table 3.2 defines the XPath fragment $\mathcal{XE} = \mathcal{XE}^{/,//,*,[]}$. Herein,

ℓ denotes any symbol from Λ, "$*$" denotes the single symbol wildcard, "." denotes the empty path expression, "/" indicates child navigation, "//" indicates descendant navigation, and "[]" indicates a predicate.

Path Language	Syntax
$\mathcal{XE} = \mathcal{XE}^{/,//,*,[]}$	$Q \rightarrow \ell \mid . \mid * \mid Q/Q \mid Q//Q \mid Q\,[Q]$

Table 3.2: The fragment $\mathcal{XE}$ of the XPath language.

The semantics of $\mathcal{XE}$ expressions as node selection queries is defined inductively as follows:

$$
\begin{aligned}
[\![\ell]\!]_T(v) &:= \{w \mid (v,w) \in E_T, lab_T(w) = \ell\} \\
[\![*]\!]_T(v) &:= \{w \mid (v,w) \in E_T\} \\
[\![.]\!]_T(v) &:= \{v\} \\
[\![Q_1/Q_2]\!]_T(v) &:= \{x \mid w \in [\![Q_1]\!]_T(v), x \in [\![Q_2]\!]_T(w)\} \\
[\![Q_1//Q_2]\!]_T(v) &:= \{y \mid w \in [\![Q_1]\!]_T(v), (x,y) \in E_T^*, y \in [\![Q_2]\!]_T(w)\} \\
[\![Q_1\,[Q_2]]\!]_T(v) &:= \{w \mid w \in [\![Q_1]\!]_T(v), [\![Q_2]\!]_T(w) \neq \emptyset\}
\end{aligned}
$$

3.5 Comparing Path ($\mathcal{PE}$) and XPath Expressions ($\mathcal{XE}$)

When comparing the path languages $\mathcal{PE}$ and $\mathcal{XE}$, it is easy to see the equivalences $\varepsilon \equiv .$ and $\_ \equiv *$ and $\_^* \equiv .//.$ and $Q_1.Q_2 \equiv Q_1/Q_2$ and $Q_1.\_^*.Q_2 \equiv Q_1//Q_2$ showing

that $\mathcal{PE}$ corresponds to $\mathcal{XE}$, while $\mathcal{PL}$ corresponds to $\mathcal{XE}^{/,//}$, and $\mathcal{PL}_s$ to $\mathcal{XE}^{/}$, See Table 3.3.

	$\mathcal{PE}$	$\mathcal{XE}$
empty path (current node)	ε	.
single wildcard	_	*
variable length wildcard	$\_^*$	.//.
child edge	$Q_1.Q_2$	Q_1/Q_2
descendant edge	$Q_1.\_^*.Q_2$	$Q_1//Q_2$

Table 3.3: Comparing $\mathcal{PE}$ and $\mathcal{XE}$.

Proposition 3.5.1. *For every expression in $\mathcal{PE}$ (or $\mathcal{PL}$, or $\mathcal{PL}_s$, respectively) there is an expression in $\mathcal{XE}$ (or $\mathcal{XE}^{/,//}$, or $\mathcal{XE}^{/}$, respectively), and vice versa, such that the respective node selection queries are equivalent.*

Example 3.5.1. *Now we can rewrite the path expressions in Example 3.3.1 in the form of $\mathcal{PE}$:*

1. *DB/Production/Movie/Actor*
2. *DB/Production/ * /Actor*
3. *//Actor*

We can see that

1. *DB.Production.Movie.Actor ≡ DB/Production/Movie/Actor*
2. *DB.Production._.Actor ≡ DB/Production/ * /Actor*
3. *_*.Actor ≡ //Actor*

Remark 3.5.1. We also can the containment relationships between those three queries above as follows:

$$\text{DB/Production/Movie/Actor} \sqsubseteq \text{DB/Production/} * \text{/Actor} \sqsubseteq \text{//Actor}$$

Chapter 4

Tree Patterns for Query Modelling

4.1 Tree Patterns

Tree patterns may be used to express node selection queries in trees, too. A *tree pattern* of arity k is a tree P with a mapping $lab_T : V_T \rightarrow \Lambda \cup \{*\}$, with a subset $E_T^{//}$ of distinguished edges, called *descendant edges*, and a k-tuple of distinguished nodes, called *output nodes*. The edges in $E_T^{/} = E_T - E_T^{//}$ are called *child edges*. Graphically, double lines represent descendant edges while single lines for child edges in a tree pattern diagram.

Definition 4.1.1. (Embedding) Consider two tree patterns P and T, We define an *embedding* to be a function $f : V_P \rightarrow V_T$ if only if the following conditions satisfied :

- *label-preserving* if $lab_P(v) = lab_T(f(v))$ or $*$ for all nodes $v \in V_P$,
- *child-preserving* if $(f(v), f(w)) \in E_T$ for all edges $(v, w) \in E_P^{/}$,
- *descendant-preserving* if $(f(v), f(w)) \in E_T^{+}$ for all edges $(v, w) \in E_P^{//}$, and

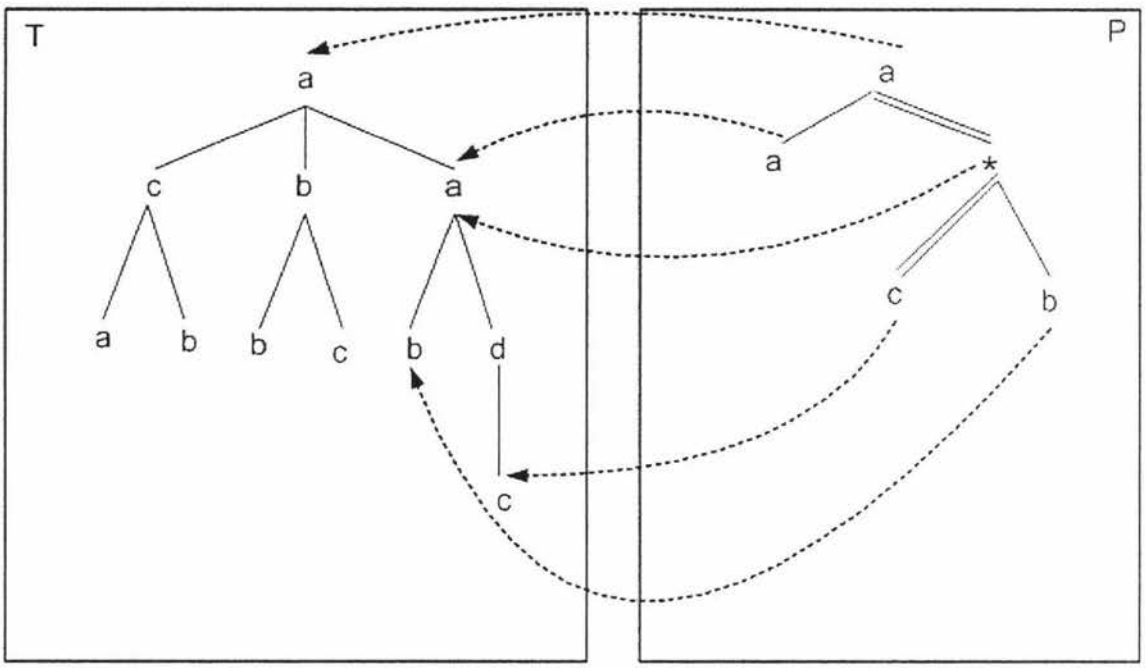

Figure 4.1: Embedding from P to T

- *root-preserving* if $lab_P(r_P) = lab_T(f(r_T))$.

Note that an embedding does not need to be injective - they may be two nodes in the pattern mapped to the same node of the tree. Even two distinct nodes may be mapped to the same node under embedding. Figure 4.1 illustrates the embedding function.

Various kinds of *tree embedding* will be discussed in Chapter 6.1.

Example 4.1.1. *Refer to the* WhereToGo *on-line ticketing example in 2.5. An $\mathcal{XE}$ expression, DB// * /Actor, can be rewritten as DB//Actor, is represented in tree patterns. They return all Actor nodes in the XML tree. The tree pattern c is represent another $\mathcal{XE}$ expression, DB/ * [Website]//Actor, which select all Actor nodes whose ancestor node has a childe node Website. See Fig 4.2*

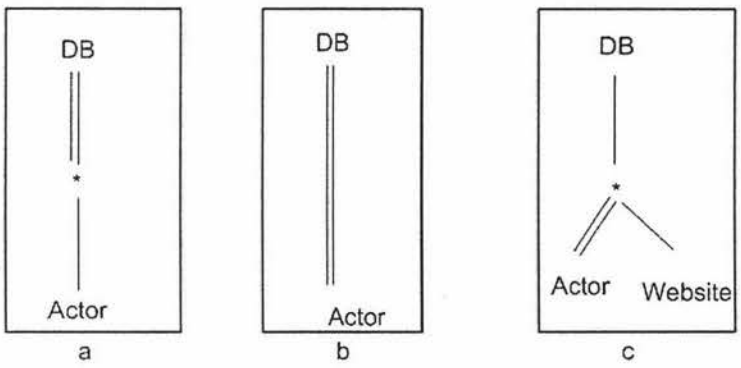

Figure 4.2: Tree pattern examples

A *homomorphism* from P' to P at node v is a label-, (*strict*) child- and descendant-preserving mapping $f : V_P \rightarrow V_T$ where $v = h(r_P)$ holds. Here we need to strengthen the child-preserving condition: if $(v, w) \in E'_{P'}$ then $(h(v), h(w)) \in E'_P$ for all edges. It does not allow to be in E''_P.

Note that a homomorphism also does not need to be injective. Figure 4.3 illustrates a homomorphism between P and P'.

Consider a *unary tree pattern* P, that is, a tree pattern of arity 1 with an output node x. The semantics of P as a node selection query may be defined as follows: $[\![P]\!]_T(v) := \{f(x) | f \text{ is a homomorphism from } P \text{ to } T \text{ at node } v\}$

Let $\mathcal{TP} = \mathcal{TP}^{/,//,*,[]}$ denote the set of all tree patterns, $\mathcal{TP}^{/,*,[]}$ the set of all tree patterns without descendant edges, $\mathcal{TP}^{/,//,[]}$ the set of all tree patterns without $*$-labels, and $\mathcal{TP}^{/,//,*}$ the set of all *linear tree patterns*, that is, tree patterns where every node has at most one child.

Miklau and Suciu [27] observed that XPath expressions can be translated into a unary tree patterns while preserving semantics. However, they request that XPath expressions (other than . itself) should only use the construct . immediately inside a predicate. Hence, an XPath expression like $a\,[.//b]$ is allowed, while $a//.$ is not. They

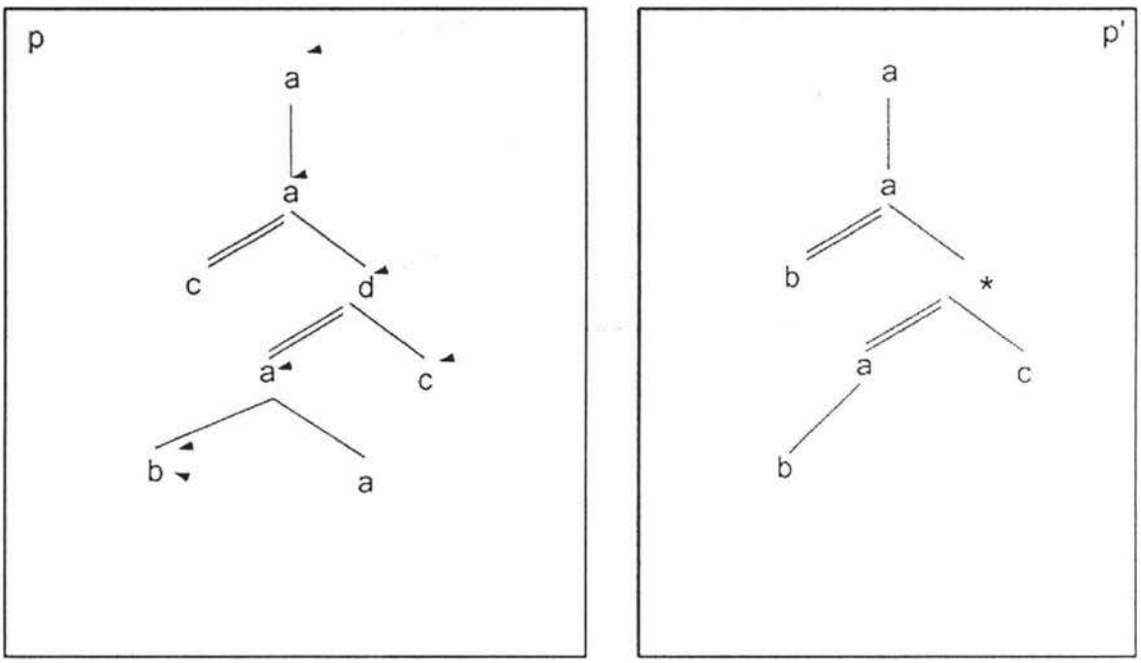

Figure 4.3: Pattern Homomorphism

argue that the semantics of $a//.$ is the union of a and $a//*$, hence the containment of $a//.$ in some expression Q can be tested by considering a and $a//*$ separately. While this remark is valid the requested restriction calls for some extra care. Let $\mathcal{XP}$ denote the set of all $\mathcal{XE}$ expressions that satisfy the requested restriction.

Miklau and Suciu further draw attention to the fact that XPath expressions ignore the label of node v at which the node selection query is executed, while tree patterns do not. For a tree pattern P let $*P$ denote the tree pattern whose root is $*$-labelled and has the root of P as its single child. Miklau and Suciu note an obvious observation similar to the following.

Proposition 4.1.1. *Let S be a subset of $\{/, //, *, []\}$ containing $/$. For every unary pattern P in $\mathcal{TP}^S$ there is an expression Q in $\mathcal{XP}^S$, and vice versa, such that the node selection queries $*P$ and Q are equivalent.*

4.2 Boolean Patterns

Tree patterns of arity 0 are called *Boolean patterns*. Every XML tree may be seen as a Boolean pattern without descendant edges. The semantics of a Boolean pattern P may be defined by setting $[\![P]\!]_T := true$ if there is a root-preserving homomorphism from P to T, and $false$ otherwise.

It should be noted that Miklau and Suciu used the term *embedding* to refer to a root-preserving homomorphism. They further refer to an algorithm by Dalvi and Sanghai that can be used to compute the value of $[\![P]\!]_T$ in quadratic time.

Proposition 4.2.1. *[27] For any Boolean pattern P and any tree T, $[\![P]\!]_T$ can be decided in time $\mathcal{O}(|P||T|)$.*

A *model* of a Boolean pattern P is a tree T on which P evaluates to true. By Mod_P we denote the set of all models of P. Two Boolean patterns P and P' are *contained*, denoted by $P \sqsubseteq P'$, if $Mod_P \subseteq Mod_{P'}$. Two Boolean patterns are *equivalent*, denoted $P \equiv P'$, by if they contain one another.

Miklau and Suciu noted that containment and equivalence problems for Boolean patterns are mutually reducible in polynomial time. Moreover, they pointed out that a solution to the containment problem for the Boolean patterns can be used to decide containment for tree patterns of higher arity. The following observations allow one to focus the investigation on Boolean patterns.

Proposition 4.2.2 (Miklau and Suciu [27])**.** *Let s be a symbol not in Λ, and let $\Lambda_0 = \Lambda \cup \{s\}$. There is a translation of unary tree patterns over Λ to Boolean patterns over Λ_0 such that for any two unary tree patterns P, P' and their translations P_0, P_0' we have that P, P' are contained if and only if P_0, P_0' are contained.*

A simple way to obtain such a translation is the following: given a unary tree pattern P, add a new s-labelled child to its output node x, and consider the derived tree pattern as a Boolean pattern P_0. Miklau and Suciu mention, however, that this approach has a disadvantage: a linear pattern P might be translated into a non-linear pattern P_0. Therefore they propose an alternative translation: given a unary tree pattern P, change the current label a of its output node to as if $a \neq *$, or to s otherwise.

Proposition 4.2.3. *[27] Let s be a symbol not in Λ, and let $\Lambda_0 = \Lambda \cup \{s, as : a \in \Lambda\}$. Let S be a subset of $\{/, //, *, []\}$ containing $/$. There is a translation of unary tree patterns over Λ to Boolean patterns over Λ_0 such that every unary pattern P in $\mathcal{TP}^S$ is translated to a Boolean pattern P_0 in $\mathcal{TP}^S$, and such that for any two unary tree patterns P, P' we have that P, P' are contained if and only if P_0, P_0' are contained.*

4.3 Canonical Models

One way to approach the containment problem for Boolean patterns is to search for a tree T that is a model for P, but not for P'. Such a tree is called a *witness*, if it exists. Hence, to decide containment it is sufficient to find a witness tree or to verify that no such witness exists. Unfortunately there can be infinitely many models for P, that is, infinitely many potential witnesses. Hence to make the approach computationally feasible one needs to reduce the search space. A promising idea is to study canonical models. They may be obtained from P in two steps: we replace descendant edges by paths of child edges, and we replace $*$-labels by some label from Λ.

Consider a Boolean pattern P, a symbol $z \in \Lambda$, and an natural number n. We

obtain the *z-substitution* $s^z(P)$ of P by substituting every $*$-label of P by a z-label. Clearly, $s^z(P[u])$ is a tree. Given a function $u : E_T \to \mathbb{N}$, we further obtain the *u-extension* $P[u]$ of P by replacing every descendant edge e of P by a path of child edges with $u(e)$ new internal nodes, labelled with $*$. The new nodes are called *extension nodes*. Moreover, we call $s^z(P[u])$ a *canonical model* for P. If $u(e) \leq n$ for all edges e of P we say that the canonical model is *n-bounded*. Let $mod^z(p)$ denote the set of all canonical models $s^z(P[u])$ for P, and $mod^z_n(p)$ the set of all n-bounded canonical models $s^z(P[u])$ for P.

Proposition 4.3.1. *[27] Let f be a root-preserving homomorphism from a Boolean pattern P to a tree T. There exists a unique extension $P[u]$ and a unique root-preserving homomorphism f' from $P[u]$ to T such that $f(v) = f'(v)$ holds for each node v of P.*

The *star length* $w(P)$ of a Boolean pattern P is the largest number of $*$-labelled nodes of P that span a path of child edges of P. According to the following observation, to decide whether P, P' are contained, it is sufficient to search for a witness tree among the $w(P')$-bounded canonical models for P. Clearly, this leaves us with a finite search space.

Proposition 4.3.2. *[27] Let P, P' be Boolean patterns, and $z \in \Lambda$ a symbol that does not occur as a label of P'. Then $P \sqsubseteq P'$ if and only if $mod^z(P) \subseteq Mod(P')$ if and only if $mod^z_{w(P')}(P) \subseteq Mod(P')$.*

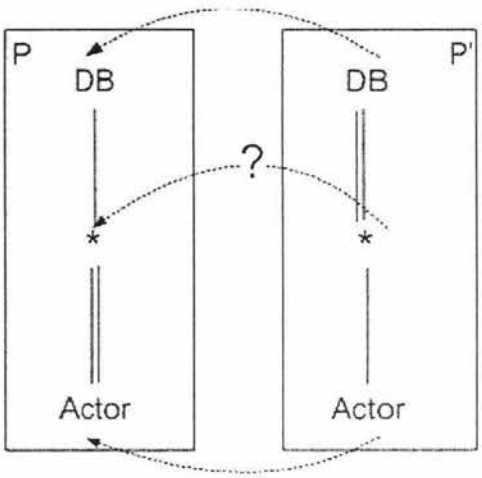

Figure 4.4: Two equivalent XPath patterns p, p' with no homomorphism

4.4 Homomorphism

Another approach to check containment is finding a homomorphism from P' to P. The existence of homomorphism between two patterns is a necessary condition for deciding containment between them, however, in some cases no homomorphism exits although $P \sqsubseteq P'$. For example, the tree patterns in Figure 4.4 corresponding to $p = DB/*//Actor$, $p' = DB//*/Actor$. Although p, p' are equivalent, there is no homomorphism from p' to p because there is no destination for the wildcard in p'. We cannot map $*$ in P' to $*$ in P because we the child edge ($*$, Actor) would not be mapped to a descendant edge.

The algorithm proposed in [27] is sound since whenever there exists a homomorphism form p' to p, them $p \sqsubseteq p'$. But, in general, the algorithm is not complete.

4.5 Deciding Query Containment

Inspecting bounded canonical models gives rise to a naive algorithm for deciding containment for Boolean patterns in time $\mathcal{O}(|P||P'|(w(P')+2)^{d(P)+1})$ where $d(P)$ denotes the number of descendent edges of P. This naive algorithm, however, is not very practical and can be improved considerably by eliminating work that is repeated for various canonical models. Miklau and Suciu then propose an algorithm that decides $P \sqsubseteq P'$ in time $\mathcal{O}(|P||P'|(w(P')+2)^{d(P)})$, and note that though the complexity of the new algorithm is only slightly better, it can run much faster in practise. Finally, by using homomorphism approach, they propose a sound, though potentially incomplete algorithm that runs in time $\mathcal{O}(|P||P'|)$. If it returns *true* then $P \sqsubseteq P'$ holds. Fortunately, the algorithm turns out to be complete in several important cases:

Theorem 4.5.1. *[27] Let P, P' be Boolean patterns such that $P \in \mathcal{TP}^{/,*,[]}$, or $P' \in \mathcal{TP}^{/,*,[]} \cup \mathcal{TP}^{/,//,[]} \cup \mathcal{TP}^{/,//,*}$. Then $P \sqsubseteq P'$ can be decided in time $\mathcal{O}(|P||P'|)$.*

In general, however, it cannot be expected that containment for Boolean patterns can be decided in polynomial time due to the next result.

Theorem 4.5.2. *[27] The problem to decide whether $P \sqsubseteq P'$, for two Boolean patterns $P \in \mathcal{TP}^{/,//,[]}$ and $P' \in \mathcal{TP}^{/,//,*,[]}$, is coNP-complete.*

Miklau and Suciu further strengthened this result to situations where $P \in \mathcal{TP}^{/,//,[]}$ and $P' \in \mathcal{TP}^{/,//,*,[]}$ has at most two $*$-labels, and where $P \in \mathcal{TP}^{/,//,[]}$ has at most three leaves and $P' \in \mathcal{TP}^{/,//,*,[]}$ has at most five leaves.

4.6 Some Known Containment Results for XPath Expressions

Using the equivalence between XPath expressions and tree patterns observed above one obtains similar results for deciding the containment of XPath expressions.

Corollary 4.6.1. *The problem to decide whether $P \sqsubseteq P'$, for two XPath expressions $P \in \mathcal{XP}^{/,//,[]}$ and $P' \in \mathcal{XP}^{/,//,*,[]}$, is coNP-complete.*

More results of XPath containment problems are summarised in Table 6.1.3

4.7 Safe Pattern Containment

In the sequel we are interested in a particular kind of tree patterns for which we are going to prove a new containment result. Recall the definition of the tree pattern. We call a tree pattern P *safe* if it contains a node v such that all descendant edges of P lie on the path from the root r_P to v, and such that no proper ancestor of v has more than one child or is an output node. Let $\mathcal{SP}$ denote the set of all safe tree patterns.

It is easy to see that $\mathcal{SP}$ includes all linear Boolean patterns.

Theorem 4.7.1. *Let P, P' be safe Boolean patterns. $P \sqsubseteq P'$ holds if and only if there is a homomorphism from P to P'.*

Proof. Clearly, $P \sqsubseteq P'$ holds whenever there exists a homomorphism from P to P'. It remains to show the converse.

To show the converse, assume there is no homomorphism from P' to P. Let $w(P')$ be the star length of P'. Consider the canonical model $t = s^z(P[u])$ of P where u

assigns every descendant edge of P the value $w(P') + 1$. For P, P' being contained, t must be a model of P', too. We will show that this is not the case, that is, we use t as a witness for P, P' being not contained. Assume there were an embedding f of P' in t. We observe that no extension node of t is the image of a node of P' under f. To the contrary suppose that f maps some node of P' to an extension node of t. Clearly, any such node of P' must be $*$-labelled. Due to the choice of t, P' must further contain a path of $w(P') + 1$ $*$-labelled nodes, or a $*$-labelled node that is incident with a descendant edge. Both cases, however, are excluded due to the definition of the star length and the definition of safe patterns, respectively. Hence, f does not map the nodes of P' to extension nodes of t, that is, f is a homomorphism from P' to P, too. But this contradicts our assumption. Therefore an embedding of P' in t does not exist, and we conclude that P, P' are not contained. This proves the statement of the theorem.

□

Corollary 4.7.2. *Let P, P' be safe Boolean patterns. $P \sqsubseteq P'$ can be decided in time $\mathcal{O}(|P||P'|)$.*

4.8 Tree Patterns of Higher Arity.

It is easy to generalise the semantics of unary tree pattern to tree patterns of arity k larger than 1. For a tree pattern P of arity $k \geq 1$ we define:

$[\![P]\!]_T(v) :=$

$\{f(x) \mid f$ is a homomorphism from P to T at node v, x is an output node of $P\}$

It should be noted that we use a different generalisation than Miklau and Suciu as this allows us to consider k-arity tree patterns as node selection queries, too. As mentioned before we can restrict most of our investigation to Boolean patterns.

Proposition 4.8.1. *Let s be a symbol not in Λ, let $\Lambda_0 = \Lambda \cup \{s\}$, and let $k \geq 1$. There is a translation of k-ary tree patterns over Λ to Boolean patterns over Λ_0 such that for any two k-ary tree patterns P, P' and their translations P_0, P_0' we have that P, P' are contained if and only if P_0, P_0' are contained.*

Such a translation may be obtained as follows: given a k-ary tree pattern P, add a new s-labelled child to each output node x, and consider the derived tree pattern as a Boolean pattern P_0. Again one might investigate more sophisticated translations. For our purposes here, however, this one is sufficient as it translates safe tree patterns into safe Boolean patterns.

Chapter 5

Tree Patterns for XML Keys

5.1 XML Keys

An *XML key* φ is an expression $(Q, Q', \{P_1, \ldots, P_k\})$ where $Q, Q', P_1, \ldots, P_k$ are node selection queries, and k is a positive integer. Herein, Q is called *context path*, Q' *target path*, and $P_1, \ldots, P_k$ *key paths*.

An XML key is called *absolute* if $[\![Q]\!]_T(r_T)$ selects just r_T, and *relative* otherwise. In another words, keys with empty context path called *absolute*, otherwise, *relative*.

An XML tree T *satisfies* φ, denoted by $T \models \varphi$, if for any node $q \in [\![Q]\!]_T(r_T)$ and for any nodes $q'_1, q'_2 \in [\![Q']\!]_T(q)$ such that there are nodes $x_i \in [\![P_i]\!]_T(q'_1)$, $y_i \in [\![P_i]\!]_T(q'_2)$ with $x_i = y_i$ for $i = 1, \ldots, k$, then $q'_1 =_v q'_2$.

More formally, we request $\forall q \in [\![Q]\!]_T(r_T)\, \forall q'_1, q'_2 \in [\![Q']\!]_T(q)$

$$\left(\bigwedge_{1 \le i \le k} [\![P_i]\!]_T(q'_1) \cap_v [\![P_i]\!]_T(q'_2) \neq \emptyset \right) \Rightarrow q'_1 = q'_2.$$

Buneman et al. [9, 10] studied XML keys where $Q, Q', P_1, \ldots, P_k$ are $\mathcal{PE}^{\cdot,-^*}$ expressions.

5.2 XML Key Patterns

It is often useful to imagine XML keys as node selection queries that return all the target nodes that are affected by the XML key. For that we build tree pattern from XML keys. Let φ be an XML key $(Q, Q', \{P_1, \ldots, P_k\})$ where $Q, Q', P_1, \ldots, P_k$ are $\mathcal{XP}$ expressions. Let K^φ be the tree pattern that corresponds to the XPath expression $Q/Q'[P_1]\cdots[P_k]$. Let q'_φ denote its output node, and q_φ denote the final node of the Q-portion of K^φ.

If we consider K^φ as a Boolean pattern we obtain the *XML key pattern* K_0^φ of φ.

In the sequel we also use two variations of this tree pattern: if we choose all leaves of K_0^φ as distinguished nodes we obtain the *leave-marked XML key pattern* K_L^φ of φ, while if we choose all descendant-or-self of q'_φ as distinguished nodes we obtain the *marked XML key pattern* K_M^φ of φ.

Example 5.2.1. *Recall the example* WhereToGo *in Chapter 2.5. We define the following keys:*

- *@Title is a key of Production node,* $(DB, (Production, \{@Title\}))$.
- *Actor is a key of Movie node,* $(.//Movie\{Actor\}))$
- *@Date and @StartTime is a key of Session node,* $(DB, (Production/Session, \{@Date, @StartTime\}))$.

Remark 5.2.1. Each key path selects a *set* of value. It allows a key to reach multiple nodes. We should note that two notions of equality are used to define keys: node identify (=), comparing two nodes in the context and target path, and value equality (val_T) when comparing nodes reached by following key paths.

In the following sections we consider the key decision problems. Basically we need to answer two questions:

- "Can we find an XML document instance satisfying a given set of key definitions?" (satisfiability problem)
- "Can we decide if a new key holds based on the a set of existing keys?" (implication problem)

5.3 Reasoning about XML keys

Consider a class $\mathcal{K}$ of XML keys. Let Σ be a finite set of XML keys in $\mathcal{K}$. An XML tree T *satisfies* Σ if and only if T satisfies every $\sigma \in \Sigma$. Let $\Sigma \cup \{\varphi\}$ be a finite set of XML keys in $\mathcal{K}$. We say that Σ *(finitely) implies* φ, denoted by $\Sigma \models_{(f)} \varphi$, if and only if every (finite) XML tree T that satisfies Σ also satisfies φ. The *(finite) implication problem* is to decide, given any finite set of XML keys $\Sigma \cup \{\varphi\}$, whether $\Sigma \models_{(f)} \sigma$. For a set Σ of keys in $\mathcal{K}$, let $\Sigma^* = \{\varphi \in \mathcal{K} \mid \Sigma \models \varphi\}$ be its *semantic closure*, that is, the set of all keys implied by Σ. Finite and unrestricted implication problem coincide for the class of XML keys in $\mathcal{K}$ [10]. We will therefore commonly speak of the *implication problem* for XML keys in $\mathcal{K}$.

The notion of derivability ($\vdash_{\mathfrak{R}}$) with respect to a set $\mathfrak{R}$ of inference rules can be defined analogously to the notion in the relational data model [1, pp. 164-168]. For a set Σ of XML keys in $\mathcal{K}$, let $\Sigma^+_{\mathfrak{R}} = \{\varphi \mid \Sigma \vdash_{\mathfrak{R}} \varphi\}$ be its *syntactic closure* under inference using $\mathfrak{R}$. The aim in this section is to find a set $\mathfrak{R}$ of inference rules which is *sound* (that is, $\Sigma^+_{\mathfrak{R}} \subseteq \Sigma^*$) and *complete* (that is, $\Sigma^* \subseteq \Sigma^+_{\mathfrak{R}}$) for the implication of XML keys. A sound and complete system of inference rules is also called an *axiomatisation*.

$$\frac{}{(Q,(\varepsilon,S))}$$ (epsilon)

$$\frac{(Q,(Q',S\cup\{\varepsilon,P\}))}{(Q,(Q',S\cup\{\varepsilon,P.P'\}))}$$ (prefix-epsilon)

$$\frac{(Q,(Q',S))}{(Q,(Q',S\cup\{P\}))}$$ (superkey)

$$\frac{(Q,(Q'.P,\{P'\}))}{(Q,(Q',\{P.P'\}))}$$ (subnodes)

$$\frac{(Q,(Q',S))}{(Q'',(Q',S))}\,Q''\sqsubseteq Q$$ (context-path-containment)

$$\frac{(Q,(Q',S))}{(Q,(Q'',S))}\,Q''\sqsubseteq Q'$$ (target-path-containment)

$$\frac{(Q,(Q'.Q'',S))}{(Q.Q',(Q'',S))}$$ (context-target)

$$\frac{(Q,(Q'.P,\{\varepsilon,P'\}))}{(Q,(Q',\{\varepsilon,P.P'\}))}$$ (subnodes-epsilon)

$$\frac{(Q,(Q',\{Q''.P_1,\ldots,Q''.P_k\})),\ (Q.Q',(Q'',\{P_1,\ldots,P_k\}))}{(Q,(Q'.Q'',\{P_1,\ldots,P_k\}))}$$ (interaction)

Table 5.1: An axiomatisation of XML keys whose context and target path are $\mathcal{PE}^{\cdot,-,*}$ expressions and whose key paths are $\mathcal{PE}^{\cdot}$ expressions.

5.4 Some Known Results for Keys with Path Expressions

Theorem 5.4.1. *[20] The system of inference rules shown in Table 5.1 is sound and complete for the implication of XML keys where Q, Q' are in $\mathcal{PE}^{\cdot,-,*}$ and $P_1,\ldots,P_k$ are in $\mathcal{PE}^{\cdot}$.*

An essential limitation of the axiomatisation is that the statement of Theorem 5.4.1 cannot be extended to XML keys whose key paths are arbitrary $\mathcal{PE}^{\cdot,-,*}$ expressions, as was observed by Hartmann and Link [20].

Proposition 5.4.2. *[20] The subnodes rule is not sound for the implication of XML keys where $Q, Q', P_1,\ldots,P_k$ are in $\mathcal{PE}^{\cdot,-,*}$.*

$$\frac{}{(Q,(.,S))}\ \text{(epsilon)} \qquad \frac{(Q,(Q',S\cup\{.,P\}))}{(Q,(Q',S\cup\{.,P/P'\}))}\ \text{(prefix-epsilon)} \qquad \frac{(Q,(Q',S))}{(Q,(Q',S\cup\{P\}))}\ \text{(superkey)}$$

$$\frac{(Q,(Q'/P,\{P'\}))}{(Q,(Q',\{P/P'\}))}\ \text{(subnodes)} \qquad \frac{(Q,(Q',S))}{(Q'',(Q',S))}\,Q''\sqsubseteq Q\ \text{(context-path-containment)} \qquad \frac{(Q,(Q',S))}{(Q,(Q'',S))}\,Q''\sqsubseteq Q'\ \text{(target-path-containment)}$$

$$\frac{(Q,(Q'/Q'',S))}{(Q/Q',(Q'',S))}\ \text{(context-target)} \qquad \frac{(Q,(Q'/P,\{.,P'\}))}{(Q,(Q',\{.,P/P'\}))}\ \text{(subnodes-epsilon)} \qquad \frac{(Q,(Q',\{Q''/P_1,\ldots,Q''/P_k\})),\ (Q/Q',(Q'',\{P_1,\ldots,P_k\}))}{(Q,(Q'/Q'',\{P_1,\ldots,P_k\}))}\ \text{(interaction)}$$

Table 5.2: A corresponding axiomatisation of XML keys whose context and target path are $\mathcal{XE}^{\{/,//\}}$ expressions and whose key paths are $\mathcal{XE}^{\{/\}}$ expressions.

5.5 Conclusions for Keys with XPath Expressions

Using the equivalence between path expressions and XPath expressions observed above one obtains similar results for XML keys with XPath expressions.

Corollary 5.5.1. *The system of inference rules shown in Table 5.1 is sound and complete for the implication of XML keys where Q, Q' are in $\mathcal{XE}^{/,//}$ and $P_1,\ldots,P_k$ are in $\mathcal{XE}^{/}$.*

Corollary 5.5.2. *The subnodes rule is not sound for the implication of XML keys where $Q, Q', P_1,\ldots,P_k$ are in $\mathcal{XE}^{/,//}$.*

5.6 Safe Key Pattern

In the sequel we want to investigate another possible extension of the statement of Theorem 5.4.1. We are interested in XML keys where Q are in $\mathcal{XE}^{/,//}$,Q' are in $\mathcal{XP}^{/,//}$

$$\frac{(Q,(Q',S\cup\{P\}))}{(Q,(Q',S\cup\{P'\}))}\,{}_{P'\sqsubseteq P}$$
key-path-containment

and $P_1,\ldots,P_k$ are in $\mathcal{XE}^{/,*}$.

Let us consider the *key-path-containment* rule. It is obvious that we need it because we define the key path as $P_1,\ldots,P_k$ are in $\mathcal{XE}^{/,*}$.

Lemma 5.6.1. *The inference rules in Table 5.1 with the* key-path-containment *rule are sound for the implication of XML keys where Q are in $\mathcal{XE}^{/,//}$,Q' are in $\mathcal{XP}^{/,//}$ and $P_1,\ldots,P_k$ are in $\mathcal{XE}^{/,*}$.*

Proof. The soundness proofs are straightforward extensions of the proofs given by Buneman et al. (for the epsilon, prefix-epsilon, superkey, context-path-containment, target-path-containment, context-target, and interaction rules), and by Hartmann and Link (for the subnodes, and subnodes-epsilon rules). As examples we formally prove the following:

(*subnodes*) Suppose an XML tree T violates the consequence $(Q,(Q',\{P/P'\}))$. Then T has a node $q\in[\![Q]\!]_T(r_T)$, and two distinct nodes $q'_1,q'_2\in[\![Q']\!]_T(q)$ such that there exist value-equal nodes $p'_1\in[\![P/P']\!]_T(q'_1)$ and $p'_2\in[\![P/P']\!]_T(q'_2)$. Clearly, T then has nodes $p_1\in[\![P]\!]_T(q'_1)$ and $p_2\in[\![P]\!]_T(q'_2)$ such that $p'_1\in[\![P']\!]_T(p_1)$ and $p'_2\in[\![P']\!]_T(p_2)$ hold. Since T is a tree and P contains no // we conclude that $p_1\neq p_2$ (since otherwise $q'_1=q'_2$). This, however, means that T violates the premise $(Q,(Q'/P,\{P'\}))$, too. We conclude that the subnodes rule is sound.

(*subnodes-epsilon*) Suppose an XML tree T violates the consequence $(Q,(Q',\{.,P/P'\}))$. Then T has a node $q\in[\![Q]\!]_T(r_T)$, and two distinct value-equal nodes $q'_1,q'_2\in[\![Q']\!]_T(q)$ such that there exist value-equal nodes $p'_1\in[\![P/P']\!]_T(q'_1)$ and $p'_2\in[\![P/$

$P']\!]_T(q_2')$. Obviously, T then has a node $p_1 \in [\![P]\!]_T(q_1')$ such that $p_1' \in [\![P']\!]_T(p_1)$. As q_1' and q_2' are value-equal there exists some node $p_2 \in [\![P]\!]_T(q_2')$ different from, but value-equal to p_1 such that $p_2' \in [\![P']\!]_T(p_2)$. But then both p_1 and p_2 belong to $[\![Q'/P]\!]_T(q)$. That is, T already violates the premise $(Q, (Q'/P, \{., P'\}))$. We conclude that the subnodes-epsilon rule is sound. □

Lemma 5.6.2. *The inference rules in Table 5.1 with the* key-path-containment *rule are complete for the implication of XML keys where Q are in $\mathcal{XE}^{/,//}$, Q' are in $\mathcal{XP}^{/,//}$ and $P_1, \ldots, P_k$ are in $\mathcal{XE}^{/,*}$.*

Let $\Sigma \cup \{\varphi\}$ be a finite set of keys. Let $\ell_0 \in \Lambda$ be a symbol that does not occur in any of key patterns of keys in $\Sigma \cup \{\varphi\}$, and let s denote the maximum star length of all key patterns of keys in $\Sigma \cup \{\varphi\}$. In the sequel we discuss how to construct an XML data tree that can serve as a counter-example for the implication of φ by Σ. We start with the canonical model $T_{\Sigma,\varphi} = mod^{\ell_0}_{s+1}(K_0^{\varphi})$ of the key pattern K_0^{φ} of φ. From $T_{\Sigma,\varphi}$ we will construct our counter-example by duplicating certain nodes. To begin with, however, we need to determine which nodes should be duplicated.

A key σ is said to be *applicable* to φ if there is a root-preserving homomorphism f from K_0^{σ} to $T_{\Sigma,\varphi}$ such that the leaves of K_0^{σ} are mapped to marked nodes of $T_{\Sigma,\varphi}$. We are particularly interested in the images $w_\sigma = f(q_\sigma)$ and $w'_\sigma = f(q'_\sigma)$ that *witness* the applicability of σ to φ. Note that there might be several such homomorphisms and thus several such witnesses.

Next we consider the *witness graph* $G_{\Sigma,\varphi}$ which we obtain from $T_{\Sigma,\varphi}$ by inserting an edge (w'_σ, w_σ) for each pair of nodes w_σ and w'_σ that witnesses the applicability of some $\sigma \in \Sigma$ to φ. We call these new edges *witness edges.*

Suppose that Σ is closed under implication, that is, $\Sigma = \Sigma^* = \Sigma^+$ holds.

Firstly we observe that for every witness edge (w'_σ, w_σ) in $G_{\Sigma,\varphi}$ and any node w between w_σ and w'_σ in $T_{\Sigma,\varphi}$ there is also a witness edge (w'_σ, w) in $G_{\Sigma,\varphi}$.

Lemma 5.6.3. *Let (w'_σ, w_σ) be a witness edge in $G_{\Sigma^+,\varphi}$. Further let w be a descendant-or-self of w_σ and an ancestor-or-self of w'_σ in $T_{\Sigma,\varphi}$. Then (w'_σ, w) is a witness edge in $G_{\Sigma^+,\varphi}$, too.*

Proof. (w'_σ, w_σ) witnesses the applicability of some $\sigma \in \Sigma^+$ to φ. We may use the *context-target* rule to obtain a suitable $\sigma' \in \Sigma^+$ from σ such that σ' is witnessed by (w'_σ, w). □

Secondly, for every witness edge (w'_σ, w_σ) in $G_{\Sigma,\varphi}$ with q'_φ between w_σ and w'_σ in $T_{\Sigma,\varphi}$ there is also a witness edge (q'_φ, w_σ) in $G_{\Sigma,\varphi}$. The latter witness edge stems from a key $\sigma' \in \Sigma^+$ whose set of key paths coincides with the set of key paths of φ.

Lemma 5.6.4. *Let (w'_σ, w_σ) be a witness edge in $G_{\Sigma^+,\varphi}$ such that q'_φ is a descendant of w_σ and an ancestor-or-self of w'_σ in $T_{\Sigma,\varphi}$. Then there is a key $\sigma' \in \Sigma^+$ that is witnessed by (q'_φ, w_σ) and whose set of key paths is the same as for as φ.*

Proof. Let (w'_σ, w_σ) witness the applicability of some $\sigma \in \Sigma = \Sigma^+$ to φ. By assumption there are $O \in \mathcal{XP}^{/,//}$ and $P \in \mathcal{XP}^{/,*}$ such that $O/P \sqsubseteq Q'_\sigma$ and such that $q'_\varphi \in [\![O]\!]_{T_{\Sigma,\varphi}}(w_\sigma)$ and $w'_\sigma \in [\![P]\!]_{T_{\Sigma,\varphi}}(q'_\varphi)$. We will show that the key $\sigma' = (Q_\sigma, (O, \{P_1^\varphi, \ldots, P_{k_\varphi}^\varphi\})) \in \Sigma = \Sigma^+$. For that we distinguish five cases.

(*Q'_σ equals .*) We conclude that $\mathcal{O}$ equals ., $w_\sigma = q'_\varphi = w'_\sigma$, and σ' may be derived from Σ by the *epsilon* rule.

(*No key path of φ equals . and P equals .*) We conclude $\mathcal{O} \sqsubseteq Q'_\sigma$ and $w'_\sigma = q'_\varphi$. Since w_σ and w'_σ witness the applicability of σ to φ we know that for each $j = 1, \ldots, k_\sigma$

there is some i with $1 \leq i \leq k_\varphi$ such that $P^\sigma_j = P^\varphi_i$. Therefore, we may infer as follows:

$$\frac{(Q_\sigma, (Q'_\sigma, \{P^\sigma_1, \ldots, P^\sigma_{k_\sigma}\}))}{\dfrac{(Q_\sigma, (O, \{P^\sigma_1, \ldots, P^\sigma_{k_\sigma}\}))}{(Q_\sigma, (O, \{P^\varphi_1, \ldots, P^\varphi_{k_\varphi}\}))}}$$

using first the *target-path-containment* rule, and then the *superkey* rule.

(*No key path of φ equals . and P differs from .*) We conclude that P is a non-empty prefix of some key path P^φ_i, and $P^\varphi_i \sqsubseteq P/P^\sigma_j$ for all $j = 1, \ldots, k_\sigma$. Since P/P^σ_j belongs to $\mathcal{XP}^{/,*}$ for all $j = 1, \ldots, k_\sigma$, there exists P', P'' in $\mathcal{XP}^{/,*}$ such that $P^\varphi_i = P'/P''$ and $P' \sqsubseteq P$ and $P'' \sqsubseteq P^\sigma_j$ for all $j = 1, \ldots, k_\sigma$. Therefore, we we may infer as follows:

$$\frac{\dfrac{\dfrac{\dfrac{\dfrac{(Q_\sigma, (Q'_\sigma, \{P^\sigma_1, \cdots, P^\sigma_{k_\sigma}\}))}{(Q_\sigma, (Q'_\sigma, \{P''\}))}}{(Q_\sigma, (O/P, \{P''\}))}}{(Q_\sigma, (O, \{P/P''\}))}}{(Q_\sigma, (O, \{P'/P''\}))}}{(Q_\sigma, (O, \{P^\varphi_1, \cdots, P^\varphi_{k_\varphi}\}))}$$

using first the *key-path-containment* rule, then the *target-path-containment* rule, then the *subnodes* rule, again the *key-path-containment* rule, and then the *superkey* rule.

(*Some key path of φ equals . and P equals .*) We conclude $\mathcal{O} \sqsubseteq Q'_\sigma$ and $w'_\sigma = q'_\varphi$. Since w_σ and w'_σ witness the applicability of σ to φ we know that for each $j = 1, \ldots, k_\sigma$ there is some P'_j in $\mathcal{XP}^{/,*}$ and some i with $1 \leq i \leq k_\varphi$ such that $P^\sigma_j/P'_j = P^\varphi_i$.

Therefore, we may infer as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(Q_\sigma, (Q'_\sigma, \{P^\sigma_1, \cdots, P^\sigma_{k_\sigma}\}))}{(Q_\sigma, (O, \{P^\sigma_1, \cdots, P^\sigma_{k_\sigma}\}))}}{(Q_\sigma, (O, \{., P^\sigma_1, \cdots, P^\sigma_{k_\sigma}\}))}}{(Q_\sigma, (O, \{., P^\sigma_1/P'_1, \cdots, P^\sigma_{k_\sigma}/P'_{k_\sigma}\}))}}{(Q_\sigma, (O, \underbrace{\{., P^\varphi_1, \cdots, P^\varphi_{k_\varphi}\}}_{=\{P^\varphi_1, \cdots, P^\varphi_{k_\varphi}\}}))}$$

using first the *target-path-containment* rule, then the *superkey* rule, then the *prefix-epsilon* rule, and then the *superkey* rule.

(*Some key path of φ equals . and P differs from .*) We conclude that P is a non-empty prefix of some key path P^φ_i. Since w_σ and w'_σ witness the applicability of σ to φ we know that for each $j = 1, \ldots, k_\sigma$ there is some P'_j in $\mathcal{XP}^{/,*}$ such that $P^\varphi_i \sqsubseteq P/P^\sigma_j/P'_j$. Since $P/P^\sigma_j/P'_j$ belongs to $\mathcal{XP}^{/,*}$ for all $j = 1, \ldots, k_\sigma$, there exists P', P'' in $\mathcal{XP}^{/,*}$ such that $P^\varphi_i = P'/P''$ and $P' \sqsubseteq P$ and $P'' \sqsubseteq P^\sigma_j/P'_j$ for all $j = 1, \ldots, k_\sigma$. Therefore, we may infer as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(Q_\sigma, (Q'_\sigma, \{P^\sigma_1, \cdots, P^\sigma_{k_\sigma}\}))}{(Q_\sigma, (O/P, \{P^\sigma_1, \cdots, P^\sigma_{k_\sigma}\}))}}{(Q_\sigma, (O/P, \{., P^\sigma_1, \cdots, P^\sigma_{k_\sigma}\}))}}{(Q_\sigma, (O/P, \{., P^\sigma_1/P'_1, \cdots, P^\sigma_{k_\sigma}/P'_{k_\sigma}\}))}}{(Q_\sigma, (O/P, \{., P''\}))}}{(Q_\sigma, (O, \{., P/P''\}))}}{(Q_\sigma, (O, \underbrace{\{., P^\varphi_1, \cdots, P^\varphi_{k_\varphi}\}}_{=\{P^\varphi_1, \cdots, P^\varphi_{k_\varphi}\}}))}$$

using first the *target-path-containment* rule, then the *superkey* rule to insert ., then

the *prefix-epsilon* rule until all P_j^σ are extended to the length of P_i^φ, then the *key-path-containment* rule, then the *subnodes-epsilon* rule, and then the *superkey* rule.

In all five cases we derived σ' from Σ which concludes the proof. □

Next we observe that if there are witness edges $(q'_\varphi, w'_{\sigma_1})$ and $(w'_{\sigma_1}, w_{\sigma_1})$ in $G_{\Sigma,\varphi}$, then there is also a witness edge $(q'_\varphi, w_{\sigma_1})$ in $G_{\Sigma,\varphi}$.

Lemma 5.6.5. *Let $(w'_{\sigma_1}, w_{\sigma_1})$ be a witness edge in $G_{\Sigma^+,\varphi}$. Further let $\sigma_2 \in \Sigma^+$ be witnessed by $(q'_\varphi, w'_{\sigma_1})$ in $G_{\Sigma^+,\varphi}$ such that its set of key paths is the same as for φ. Then there is some $\sigma' \in \Sigma^+$ that is witnessed by $(q'_\varphi, w_{\sigma_1})$ in $G_{\Sigma^+,\varphi}$ and whose set of key paths is the same as for φ, too.*

Proof. Let $(w'_{\sigma_1}, w_{\sigma_1})$ witness the applicability of a key $\sigma_1 \in \Sigma^+$ to φ. By assumption there is some P in $\mathcal{XP}^{/,*}$ such that for each $j = 1, \ldots, k_{\sigma_1}$ there exists $O_j^{\sigma_1}$ in $\mathcal{XP}^{/,*}$ with $P/O_j^{\sigma_1} \sqsubseteq P_j^{\sigma_1}$, $P \sqsubseteq Q'_{\sigma_2}$, and $q'_\varphi \in [\![P]\!]_{T_{\Sigma,\varphi}}(w'_{\sigma_1})$. Therefore, we may infer as follows:

$$\frac{(Q_{\sigma_1}, (Q'_{\sigma_1}, \{P_1^{\sigma_1}, \cdots, P_{k_{\sigma_1}}^{\sigma_1}\}))}{(Q_{\sigma_1}, (Q'_{\sigma_1}, \{P/O_1^{\sigma_1}, \cdots, P/O_{k_{\sigma_1}}^{\sigma_1}\}))}$$
$$\overline{(Q_{\sigma_1}, (Q'_{\sigma_1}, \{P/P_1^{\varphi}, \cdots, P/P_{k_\varphi}^{\varphi}, P/O_1^{\sigma_1}, \cdots, P/O_{k_{\sigma_1}}^{\sigma_1}\}))}$$

using first the *key-path-containment* rule, and then the *superkey* rule.

By assumption we further have $Q_{\sigma_1}/Q'_{\sigma_1} \sqsubseteq Q_{\sigma_2}$. We may infer as follows:

$$\frac{(Q_{\sigma_2}, (Q'_{\sigma_2}, \{P_1^{\varphi}, \cdots, P_{k_\varphi}^{\varphi}\}))}{(Q_{\sigma_1}/Q'_{\sigma_1}, (P, \{P_1^{\varphi}, \cdots, P_{k_\varphi}^{\varphi}, O_1^{\sigma_1}, \cdots, O_{k_{\sigma_1}}^{\sigma_1}\}))}$$

using the *context-path-containment*, *target-path-containment* and *superkey* rules. Together we derive

$$(Q_{\sigma_1}, (Q'_{\sigma_1}/P, (P, \{P_1^{\varphi}, \cdots, P_{k_\varphi}^{\varphi}, O_1^{\sigma_1}, \cdots, O_{k_{\sigma_1}}^{\sigma_1}\})) \in \Sigma^+$$

using the *interaction* rule.

Recall that σ_1 is applicable to φ. Suppose that no P_i^{φ}, $i = 1, \ldots, k_{\varphi}$ equals ., then we find that for each $j = 1, \ldots, k_{\sigma_1}$ there is some i with $1 \leq i \leq k_{\varphi}$ such that $P_i^{\varphi} \sqsubseteq P_j^{\sigma_1}$. Hence we conclude

$$(Q_{\sigma_1}, (Q'_{\sigma_1}/P, \{P_1^{\varphi}, \cdots, P_{k_{\varphi}}^{\varphi}\})) \in \Sigma^+.$$

Otherwise, for all $j = 1, \ldots, k_{\sigma_1}$ there is some P'_j in $\mathcal{XP}^{/,*}$ and some i with $1 \leq i \leq k_{\varphi}$ such that $P_i^{\varphi} \sqsubseteq O_j^{\sigma_1}/P'_j$. Hence we obtain

$$(Q_{\sigma_1}, (Q'_{\sigma_1}/P, \{P_1^{\varphi}, \cdots, P_{k_{\varphi}}^{\varphi}, O_1^{\sigma_1}/P'_1, \cdots, O_{k_{\sigma_1}}^{\sigma_1}/P'_{k_{\sigma_1}}\})) \in \Sigma^+$$

using the *prefix-epsilon* rule, and then the *key-path-containment* rule yields

$$(Q_{\sigma_1}, (Q'_{\sigma_1}/P, \{P_1^{\varphi}, \cdots, P_{k_{\varphi}}^{\varphi}\})) \in \Sigma^+.$$

This verifies $\sigma_2 \in \Sigma^+$ and concludes the proof. □

Lemma 5.6.6. *Let $(q'_{\varphi}, q_{\varphi})$ be in the reflexive and transitive closure of the witness graph $G_{\Sigma,\varphi}$. Then $\varphi \in \Sigma^+$.*

Proof. If $(q'_{\varphi}, q_{\varphi})$ is in the reflexive and transitive closure of the witness graph $G_{\Sigma,\varphi}$ then it is also in the reflexive and transitive closure of the witness graph $G_{\Sigma^+,\varphi}$. Hence we may assume that Σ is closed, that is, $\Sigma = \Sigma^+$.

Since $(q'_{\varphi}, q_{\varphi})$ is in the reflexive and transitive closure of the witness graph $G_{\Sigma^+,\varphi}$ we have a path from q'_{φ} to q_{φ} in $G_{\Sigma^+,\varphi}$. By Lemma 5.6.3 we obtain a path from a descendant-or-self of q'_{φ} to q_{φ} in $G_{\Sigma^+,\varphi}$ that consists only of witness edges. By Lemma 5.6.4 we obtain a path from q'_{φ} to q_{φ} in $G_{\Sigma^+,\varphi}$ such that the first edge on the path witnesses the applicability of some key $\sigma' \in \Sigma^+$ that has the same set of key paths as

φ. Repeated application of Lemma 5.6.5 then shows that (q'_φ, q_φ) is an edge in $G_{\Sigma^+,\varphi}$ that witnesses the applicability of some key $\sigma \in \Sigma^+$ having the same set of key paths as φ. Since σ is applicable to φ we conclude $Q_\varphi \sqsubseteq Q_\sigma$ and $Q'_\varphi \sqsubseteq Q'_\sigma$. Using the *context-path-containment* and the *target-path-containment* rules allows us to derive φ from σ. Hence $\varphi \in \Sigma^+$.

□

5.7 Time Complexity of Deciding Implication

In this section we first use the equivalence between XPath expressions $\mathcal{XE}$ and tree patterns $\mathcal{TP}$ to obtain similar results for deciding implication of *Safe Key Pattern*.

Recall Theorem 4.7.1, the inference rules in Table 5.1, Lemma 5.6.1 and Lemma 5.6.2. We conclude the following theorem:

Theorem 5.7.1. *The implication of two safe key patterns can be decided in time* $\mathcal{O}(|P||P'|)$.

The next step is to use the technique of proving completeness to develop an efficient algorithm for deciding XML key implication in form of *safe tree pattern*. However, it is beyond the scope of this thesis.

Chapter 6

Conclusion

6.1 Related Work

Tree matching problem has many applications. XML is one of them. In the following three sections we review the tree matching problem in general.

6.1.1 Tree Embeddings

In the literature various kinds of tree embeddings have been defined and studied. Consider two trees P and T. We are interested whether P can be embedded into T. Generally, an embedding of P into T is a mapping $f : V_P \rightarrow V_T$ from the node set V_P to the node set V_T that satisfies certain properties. In Table 6.1 we list a number of desirable properties of tree embeddings.

Clearly, the last two properties are only applicable if the tree are labelled or ordered, respectively.

Consider a mapping $f : V_P \rightarrow V_T$ which is one-to-one, label-preserving (if P and T

Property of $f : V_P \rightarrow V_T$	Conditions for any nodes a and b of P
one-to-one	$f(a) \neq f(b)$ if $a \neq b$
root-to-root	$f(a)$ is a root if a is a root
descendant preserving	$(a, b) \in E_P^{//}$ if $(f(a), f(b)) \in E_T^+$
strict descendant preserving	$(a, b) \in E_P^{//}$ if only if $(f(a), f(b)) \in E_T^+$
child-preserving	$(a, b) \in E_P^{/}$ if $(f(a), f(b)) \in E_T$
strict child-preserving	$(a, b) \in E_P^{/}$ if only if $(f(a), f(b)) \in E_T$
label-preserving	$a \in V_P$ if $lab_P(a) = lab_T(f(a))$ or $*$
order-preserving	$f(a) \succ f(b)$ if $a \succ b$

Table 6.1: Properties of tree embeddings

are labelled). In the literature, f is called a *tree inclusion* when it is strict descendant preserving. f is called *subgraph isomorphism* embedding when it is restrict child-preserving. f is called a *topological embedding* when it is descendant preserving. and for any node v in P with distinct children a and b, $f(v)$ is *the smallest common ancestor* of $f(a)$ and $f(b)$ in T.

Also, f is an *isomorphism* when it is onto and strict descendant preserving. In the case, P and T are isomorphism copied of one another that only differ by the underlying node sets. Normally we do not distinguish between isomorphic copies, that is, we consider two tress to be the *same* if there is an isomorphism between them.

6.1.2 Tree Matching Problems

A general introduction of tree matching problems can be found in [25] by Kilpeläinen. Bille [6] summarised the problems of labelled tree matching based on simple primitive operations of *relabelling*, *inserting* and *deleting*. Let T_1 and T_2 be labeled trees (ordered and unordered). Based on the edit operations, Bille surveyed each of the *tree edit*

distance, *alignment* and *inclusion* problem and discussed the results obtained from them respectively. See Table 6.2 below for reference.

Tree Edit Distance

For each edit operation a *cost function* is defined. An *edit script* S between T_1 and T_2 is a sequence of edit operations transforming T_1 to T_2. The cost of S is the sum of the costs of the operation in S. An *optimal edit script* between T_1 and T_2 is an edit script between T_1 and T_2 of minimum cost and this cost is the *tree edit distance*. The tree edit distance problem is to compute the edit distance and a corresponding edit script.

Definition 6.1.1. **(tree edit distance)** Given a metric cost function γ defined on pairs of labels, the cost of an edit operation $\gamma(l_1 \rightarrow l_2)$=$\gamma(l_1, l_2)$ The cost of a sequence S is given by $\gamma(S) = \sum_{i=1}^{k} \gamma(S_i)$. The edit distance, denoted by $\delta(T_1, T_2)$, between T_1, T_2 is formally defined as follows:

$$\delta(T_1, T_2) = \min\{\gamma(S) | \text{ S is a sequence of operations transforming } T_1 \text{ into } T_2\}$$

The edit distance problem is the most intensively studied topic [37, 38, 30, 33]. Recently it has gained increased attentions of XML database community because XML documents are viewed as rooted, and labelled trees. The results in Table 6.2 of ordered trees directly contribute toward to XML research. Various algorithms for determining structural similarity between XML documents have been proposed. Recent work [32] by Tekli used the traditional tree edit distance approach with the improvement of taking the structural commonality between subtrees into consideration. This approach first searches for the similarity of two subtrees and then compute tree edit operations cost. The overall complexity of this methods is linear in the number of

nodes of each tree, and polynomial (quadratic) in the size of the two trees being compared: $\mathcal{O}(|T_1||T_2|)$, which can be simplified to $\mathcal{O}(N^2)$, N being the maximum number of nodes in T_1 and T_2. The idea maybe be relevant to the XML key implication problems we are interested in.

Tree Alignment Distance

Another measure of similarity between two trees is tree alignment distance [23]. Bille [6] argued that the tree alignment distance, denoted as $\alpha(T_1, T_2)$, is a special case of the tree edit distance problem: all insertion must be performed before deletions.. Hence, $\delta(T_1, T_2) \leq \alpha(T_1, T_2)$.

Tree Inclusion Problems

This problem has recently been recognized as an important query primitive in XML databases. Bille [6] defined the tree inclusion problem as follows: Given two rooted and labeled trees P and T the tree inclusion problem is to determine if P can be obtained from T by deleting nodes in T. Most of the results of the tree inclusion problem shown in Table 6.2 are improved from the first polynomial (quadratic) time algorithm [26] by Kilpeläinen and Mannila. In [7] Bille and Inge Li Gørtz took a different approach to the problem which leads to a new algorithm which uses optimal linear space and has subquadratic running time.

Here we need to point out that tree inclusion problem is equivalent to finding homomorphism between two trees. As we discuss in Chapter 4, homomorphism is a useful technique to discover containment. but for some cases of containment there is no homomorphism.

6.1.3 Computational Complexity of Tree Edit Distance, Alignment Distance and Inclusion

In Table 6.2 [6] most of the available results of tree matching problems are summarised. Some definitions are required here. Consider T_i is a tree, D_i denotes the depth of the tree T_i, L_i means the number of leaves, and I_i is for the maximum degree of T_i. The type is either O for ordered or U for unordered. The value u is the *unit cost edit distance* between T_1 and T_2 and the value s is the number of spaces in the optimal alignment of T_1 and T_2. The value Σ_{T_i} is set of labels used in T_1 and m_{T_1,T_2} is the number of pairs of nodes in T_1 and T_2 which have the same labels.

Computational Complexity of XPath Containment

Some more results on the complexity of the XPath containment problem can be found in [36, 28, 27, 4]. Most them focus on the containment problem for a simple fragment of XPath which is used frequently in practice. The definition of the fragment $\mathcal{XE}$ of the XPath language can be found in Table 3.2 of Chapter 3.4. In [28], in addition to the fragment $\mathcal{XE}$, Neven and Schwentick consider disjunction, DTDs and variables. Computational complexity of containment problems in the context of XPath in the presence of those three factors are summarised in Table 6.3. Note "|" denotes disjunction.

6.1.4 Integrity Constraints for XML

In this section we identify some specifically related work focusing on integrity constraints for XML. For a brief survey on recent work on XML constraints see [15].

Numerical Keys

Hartmann and Link [19] introduced numerical constraints and generalised the key definition in [9, 10] into numerical key. The idea of numerical constraints is from cardinality and participation constraints in conceptual database design. A numerical key has been defined as follows: $card(Q, (Q', P_1, ..., P_k)) \leq b$. Q, Q' and P_k are same with paths as defined in [9, 10] and b is a positive integer or ∞. Therefor, the key definition in [9, 10] is a special case of the numerical key constraint when $b = 1$. Hartmann and Link have proved that each finite set of numerical keys is finitely satisfiable and the implication and finite implication problem for numerical keys coincide. A set of axiomatisation rules for numerical key have been investigated. Consequently, they have proposed an algorithm of deciding implication. The run time of the algorithm is in quadratic time. In another words, the expressiveness of numerical keys is increased but the time complexity of their related implication problems remains same. In addition, Hartmann and Link showed in [21] that efficient reasoning about the numerical constraints provides means to predict query costs and results, therefore,helps XQuery optimisation. Other applications include updating, encrypting and indexing .

Techniques for Reasoning about XML Keys

The implication problem for key constraints is harder than in the relational case simply because it involves reasoning about path expression, i.e., path containment problem. See [29] for a survey on the techniques used to obtain containment results for various fragments of XPath. In the section we summarise the different techniques used in literature for reasoning about XML keys .

The technique used in [9, 10] is *chase*. Generally speaking, it works as follows: for any $\varphi \notin \Sigma^+$, a finite XML tree T is constructed. T violates φ. Then they *chase* those keys in Σ which are violated by T and maintain at the same time the violation of φ. Finally, this results into a chased version of T which finitely satisfies all keys in Σ and violates φ. This would show that φ is not implied by Σ.

However, The work [20] by Hartmann and Link shows that one of the inference rules for key implication in [9, 10] is only sound for keys with simple key path expression, but not sound in general as sated. A set of sound and complete inference rules have been proposed for the implication of XML keys with simple key path in [20]. The technique used in [20] is based on *diagraph*. It contains two main steps: first, construct a witness graph $G_{\Sigma,\varphi}$; second, keep track of a reachability graph. if reachability exits then Σ implies, otherwise no. In fact, the key φ is implied by Σ iff there is a dipath from the existence of reachability in $G_{\Sigma,\varphi}$.

From this we can see the importance of choosing the reasoning techniques.

In [19] the technique above using *diagraph* has been extended and adopted shortest path methods of *Aho-Corasick automata* for the construction of cardinality graphs.

6.2 Conclusion

In this thesis we have reviewed the key constraint languages for XML. We have examined three alternative approaches of expressing node selection queries, studied translation and discovered equivalences between them. Three path expression languages are path expression $\mathcal{PE}$, XPath expression $\mathcal{XE}$ and tree pattern $\mathcal{TP}$. Our conclusions are as follows:

- For every expression in $\mathcal{PE}$ (or $\mathcal{PL}$, or $\mathcal{PL}_s$, respectively) there is an expression in $\mathcal{XE}$ (or $\mathcal{XE}^{/,//}$, or $\mathcal{XE}^{/}$, respectively), and vice versa, such that the respective node selection queries are equivalent.

- Let S be a subset of $\{/, //, *, []\}$ containing $/$. For every unary pattern P in $\mathcal{TP}^S$ there is an expression Q in $\mathcal{XP}^S$, and vice versa, such that the node selection queries $*P$ and Q are equivalent.

we have investigated the implication problem of XML keys which is defined in a special case of tree patterns, namely, the *safe tree pattern* . In the first part of our work some results have been obtained with respect to it:

1. We defined *safe tree pattern.* We call a tree pattern P *safe* if it contains a node v such that all descendant edges of P lie on the path from the root r_P to v, and such that no proper ancestor of v has more than one child or is an output node.

2. We discovered equivalences between it and other node selection queries.

3. We reasoned about its containment problem for which we can prove a new containment result.

4. We investigated its computational complexity. The time complexity is in PTIME.

In the second part of our work, in accordance with other work relating to XML keys, we have proved a sound and complete axiomatisation of XML keys using *safe tree pattern* as path expression. It is a generalised format of XML key definition compared with the one in [9, 10]. Descendant edges (//) are only allowed to appear in the context path and target path, and wildcards ($*$) are allowed in the key path. The containment problem for the *safe tree pattern* is in PTIME, hence, the implication

problem of XML keys defined in the form the *safe tree pattern* can be decided in PTIME. Therefore, the XML key can be maintained efficiently by database systems for XML applications.

6.3 Future Work

There are several areas of research which is still need to be investigated to proceed towards developing a sound and complete theory of XML keys.

In Chapter 5 we have observes that the time complexity of the implication problem of the XML key patterns in form of *safe tree pattern* is in PTIME. In accordance with other work deciding implication of XML keys, our immediate work is to develop and implement a sound and complete algorithm to check implication automatically and efficiently.

Although compared with the definition in [9, 10, 20], the key pattern defined in this thesis is a generalized version. However, for XML keys with arbitrary key paths the axiomatisation problem, consequently, the satisfiability and implication problem, are still open. We need to fully address this problem in order to get a generalised XML key definition. Another research direction is to investigate the feasibility of using other semantics for XML keys and associated decision problems.

The long term research directions include investigations of interaction between constraints and schema definition languages (DTD or XSD). From the previous work we can see there is an intricate interaction between XML key constraints and schema specifications. The numerical constraints introduced in [19] suggests that generalisation of XML key constraints and several classes cardinality constraints studies could be a new research interest. Taking other constraints, such as functional dependency,

[22], into consideration, a unified XML constraint system could be developed and utilised effectively by XML applications. Here future work will focus on identifying useful restricted classes of XML constraints.

Table 6.2: Results for tree edit distance, alignment distance and inclusion problems

Tree edit distance

variant	type	time	space
general	O	$\mathcal{O}(\|T_1\|\|T_2\|D_1^2D_2^2)$	$\mathcal{O}(\|T_1\|\|T_2\|D_1^2D_2^2)$
general	O	$\mathcal{O}(\|T_1\|\|T_2\|min(L_1,D_1)min(L_2,D_2))$	$\mathcal{O}(\|T_1\|\|T_2\|)$
general	O	$\mathcal{O}(\|T_1\|^2\|T_2\|\log\|T_2\|)$	$\mathcal{O}(\|T_1\|\|T_2\|)$
general	O	$\mathcal{O}(\|T_1\|\|T_2\| + L_1^2\|T_2\| + L_1^{2.5}L_2)$	$\mathcal{O}((\|T_1\| + L_1^2) + min(L_2,D_2) + \|T_2\|)$
general	U	MAX SNP-hard	
constrainted	O	$\mathcal{O}(\|T_1\|\|T_2\|)$	$\mathcal{O}(\|T_1\|\|T_2\|)$
constrainted	O	$\mathcal{O}(\|T_1\|\|T_2\|I_1I_2)$	$\mathcal{O}(\|T_1\|D_2I_2)$
constrainted	U	$\mathcal{O}(\|T_1\|\|T_2\|(I_1+I_2)\log(I_1+I_2))$	$\mathcal{O}(\|T_1\|\|T_2\|)$
less-constrainted	O	$\mathcal{O}(\|T_1\|\|T_2\|I_1^3I_2^3(I_1+I_2))$	$\mathcal{O}(\|T_1\|\|T_2\|I_1^3I_2^3(I_1+I_2))$
less-constrainted	U	MAX SNP-hard	
unit-cost	O	$\mathcal{O}(u^2min(\|T_1\|,\|T_2\|)min(L_1,L_2)$	$\mathcal{O}(\|T_1\|\|T_2\|)$
1-degree	O	$\mathcal{O}(\|T_1\|\|T_2\|)$	$\mathcal{O}(\|T_1\|\|T_2\|)$

Tree alignment distance

variant	type	time	space
general	O	$\mathcal{O}(\|T_1\|\|T_2\|(I_1+I_2)^2)$	$\mathcal{O}(\|T_1\|\|T_2\|(I_1+I_2)^2)$
general	O	$\mathcal{O}((\|T_1\|+\|T_2\|)\log(\|T_1\|+\|T_2\|)(I_1+I_2)^4s^2))$	$\mathcal{O}((\|T_1\|+\|T_2\|)\log(\|T_1\|+\|T_2\|)(I_1+I_2)^4s^2))$
general	U	MAX SNP-hard	

Tree inclusion

variant	type	time	space
general	O	$\mathcal{O}(\|T_1\|\|T_2\|)$	$\mathcal{O}(\|T_1\|min(D_2L_2))$
general	O	$\mathcal{O}(\|\Sigma_{T_1}\|\|T_2\| + m_{T_1,T_2}D_2)$	$\mathcal{O}(\|\Sigma_{T_1}\|\|T_2\| + m_{T_1,T_2})$
general	O	$\mathcal{O}(L_1\|T_2\|)$	$\mathcal{O}(L_1min(D_2L_2))$
general	U	NP-hard	

Time	Classifications
PTIME	$\mathcal{XP}^{/,//}$, $\mathcal{XP}^{/,//,[]}$, $\mathcal{XP}^{/,*,[]}$, $\mathcal{XP}^{/,//,*,[]}$, $\mathcal{XP}^{/,//,*,DTD}$
coNP-complete	$\mathcal{XP}^{/,//,*,[]}$, $\mathcal{XP}^{/,//,*,[],\mid}$, $\mathcal{XP}^{/,[],DTD}$
coNP-hard	$\mathcal{XP}^{/,\mid}$, $\mathcal{XP}^{//,\mid}$, $\mathcal{XP}^{//,[],\mid,DTD}$
EXPTIME-complete	$\mathcal{XP}^{/,//,*,\mid,[],DTD}$
EXPTIME-complete	$\mathcal{XP}^{/,//,*,[],DTD}$, $\mathcal{XP}^{/,//,\mid,[],DTD}$

Table 6.3: The path containment problem

Bibliography

[1] Serge Abiteboul, Richard Hull, and Victor Vianu, *Foundations of databases: The logical level*, Addison Wesley; Facsimile edition (13 Feb 1995).

[2] Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Ian Jacobs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Robert Sutor, Chris Wilson, and Lauren Wood., *Document object model (dom) level specification. w3c recommendation, october 1998.*, The World Wide Web Consortium (W3C), October 1998.

[3] Marcelo Arenas, Wenfei Fan, and Leonid Libkin, *What's hard about xml schema constraints?*, DEXA, 2002, pp. 269–278.

[4] Michael Benedikt, Wenfei Fan, and Gabriel M. Kuper, *Structural properties of xpath fragments*, in Calvanese et al. [11], pp. 79–95.

[5] Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, and Jérôme Siméon, *Xml path language (xpath) 2.0 w3c recommendation 23 january 2007*, The World Wide Web Consortium (W3C), January 2007.

[6] Philip Bille, *A survey on tree edit distance and related problems*, Theor. Comput. Sci. **337** (2005), no. 1-3, 217–239.

[7] Philip Bille and Inge Li Gørtz, *The tree inclusion problem: In optimal space and faster*, ICALP (Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia

Palamidessi, and Moti Yung, eds.), Lecture Notes in Computer Science, vol. 3580, Springer, 2005, pp. 66–77.

[8] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau eds, *Extensible markup language (xml) 1.0 (fourth edition) w3c recommendation 16 august 2006*, The World Wide Web Consortium (W3C), August 2006.

[9] Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan, *Keys for xml*, Computer Networks **39** (2002), no. 5, 473–487.

[10] ———, *Reasoning about keys for xml*, Inf. Syst. **28** (2003), no. 8, 1037–1063.

[11] Diego Calvanese, Maurizio Lenzerini, and Rajeev Motwani (eds.), *Database theory - icdt 2003, 9th international conference, siena, italy, january 8-10, 2003, proceedings*, Lecture Notes in Computer Science, vol. 2572, Springer, 2002.

[12] Stefano Ceri, Piero Fraternali, and Stefano Paraboschi, *Xml: Current developments and future challenges for the database community*, EDBT, 2000, pp. 3–17.

[13] Charles F., *The sgml handbook*, Oxford University Press, 1990.

[14] David C. Fallside and Priscilla Walmsley, *Xml schema part 0: Primer second edition w3c recommendation 28 october 2004*, The World Wide Web Consortium (W3C), October 2004.

[15] Wenfei Fan, *Xml constraints: Specification, analysis, and applications*, DEXA Workshops, IEEE Computer Society, 2005, pp. 805–809.

[16] Wenfei Fan, Gabriel M. Kuper, and Jérôme Siméon, *A unified constraint model for xml*, Computer Networks **39** (2002), no. 5, 489–505.

[17] Wenfei Fan and Leonid Libkin, *On xml integrity constraints in the presence of dtds*, J. ACM **49** (2002), no. 3, 368–406.

[18] Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, and Norman Walsh, *Xquery 1.0 and xpath 2.0 data model (xdm) w3c recommendation 23 january 2007*, The World Wide Web Consortium (W3C), January 2007.

[19] Sven Hartmann and Sebastian Link, *Numerical constraints for xml*, WoLLIC (Daniel Leivant and Ruy J. G. B. de Queiroz, eds.), Lecture Notes in Computer Science, vol. 4576, Springer, 2007, pp. 203–217.

[20] ———, *Unlocking keys for xml trees*, ICDT, 2007, pp. 104–118.

[21] ———, *Xml query optimisation: Specify your selectivity*, DEXA Workshops, IEEE Computer Society, 2007, pp. 30–34.

[22] Sven Hartmann and Thu Trinh, *Axiomatising functional dependencies for xml with frequencies*, FoIKS (Jürgen Dix and Stephen J. Hegner, eds.), Lecture Notes in Computer Science, vol. 3861, Springer, 2006, pp. 159–178.

[23] Tao Jiang, Lusheng Wang, and Kaizhong Zhang, *Alignment of trees - an alternative to tree edit*, Theor. Comput. Sci. **143** (1995), no. 1, 137–148.

[24] Michael Kay, *Xsl transformations (xslt) version 2.0 w3c recommendation 23 january 2007*, The World Wide Web Consortium (W3C), January 2007.

[25] Pekka Kilpeläinen, *Tree matching problems with applications to structured text databases*, Phd thesis, Department of Computer Science, University of Helsinki, 1992.

[26] Pekka Kilpeläinen and Heikki Mannila, *Ordered and unordered tree inclusion*, SIAM J. Comput. **24** (1995), no. 2, 340–356.

[27] Gerome Miklau and Dan Suciu, *Containment and equivalence for a fragment of xpath*, J. ACM **51** (2004), no. 1, 2–45.

[28] Frank Neven and Thomas Schwentick, *Xpath containment in the presence of disjunction, dtds, and variables*, in Calvanese et al. [11], pp. 312–326.

[29] Thomas Schwentick, *XPath query containment*, SIGMOD Record **33** (2004), no. 1, 101–109.

[30] Dennis Shasha, Jason Tsong-Li Wang, Huiyuan Shan, and Kaizhong Zhang, *Atreegrep: Approximate searching in unordered trees*, SSDBM, IEEE Computer Society, 2002, pp. 89–98.

[31] Dan Suciu, *On database theory and xml*, SIGMOD Rec. **30** (2001), no. 3, 39–45.

[32] Joe Tekli, Richard Chbeir, and Kokou Yétongnon, *A fine-grained xml structural comparison approach*, ER (Christine Parent, Klaus-Dieter Schewe, Veda C. Storey, and Bernhard Thalheim, eds.), Lecture Notes in Computer Science, vol. 4801, Springer, 2007, pp. 582–598.

[33] Hélène Touzet, *Tree edit distance with gaps*, Inf. Process. Lett. **85** (2003), no. 3, 123–129.

[34] Victor Vianu, *A web odyssey: from codd to xml*, SIGMOD Rec. **32** (2003), no. 2, 68–77.

[35] Jennifer Widom, *Data management for xml: Research directions*, IEEE Data Eng. Bull. **22** (1999), no. 3, 44–52.

[36] Peter T. Wood, *Containment for xpath fragments under dtd constraints*, in Calvanese et al. [11], pp. 297–311.

[37] Kaizhong Zhang and Dennis Shasha, *Simple fast algorithms for the editing distance between trees and related problems*, SIAM J. Comput. **18** (1989), no. 6, 1245–1262.

[38] Kaizhong Zhang, Richard Statman, and Dennis Shasha, *On the editing distance between unordered labeled trees*, Inf. Process. Lett. **42** (1992), no. 3, 133–139.