Vergleich von Ansätzen wertbasierter Integritätsbedingungen in XML

Diplomarbeit

zur Erlangung des akademischen Grades eines Magisters
der Sozial- und Wirtschaftswissenschaften
im Diplomstudium Wirtschaftsinformatik

Eingereicht an der Johannes Kepler Universität Linz
Institut für Wirtschaftsinformatik
Data & Knowledge Engineering

Eingereicht bei: o. Univ.-Prof. Dr. Michael Schrefl
Betreuender Assistent: Dr. Michael Karlinger

Manuela Schrattenecker

Linz, Januar 2014

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe. Die vorliegende Diplomarbeit ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, Januar 2014

Manuela Schrattenecker

Kurzfassung

XML entwickelte sich im Laufe der letzten Jahre zu einem der wichtigsten Formate für den Datenaustausch. Um übermittelte XML Daten verarbeiten zu können ist es entscheidend, die Struktur und Semantik der Daten zu kennen. Für das Spezifizieren der Struktur von XML Daten konnten sich die vom World Wide Web Consortium entwickelten Schemasprachen XML Schema Definition und Document Type Definition als Standards etablieren. Im Unterschied dazu konnte sich bisher keine Herangehensweise für das Spezifizieren der Semantik von XML Daten mittels Integritätsbedingungen durchsetzen. Die in der Fachliteratur vorgestellten XML Integritätsbedingungen unterscheiden sich teils grundlegend voneinander, weisen aber auch Gemeinsamkeiten auf.

Ziel dieser Diplomarbeit ist es, diese Gemeinsamkeiten in Form einer Liste von Vergleichsmerkmalen heraus zu arbeiten, um dadurch eine einheitliche Vergleichsbasis für Ansätze wertbasierter XML Integritätsbedingungen zu schaffen. Anhand der erarbeiteten Vergleichsmerkmale werden einige sehr prominente Ansätze analysiert und einander abschließend gegenübergestellt.

Abstract

During the last few years, XML became one of the most important formats for data exchange over the Internet. In order to process transmitted data it is essential to know their structure and semantics. For specifying the structure of XML data, the schema languages XML Schema Definition and Document Type Definition, both developed by the World Wide Web Consortium, have become standards. In contrast, none of the approaches for specifying the semantics of XML data by means of integrity constraints have become prevalent so far. The XML integrity constraints presented in literature differ in many respects, but do also have certain commonalities.

The purpose of this diploma thesis is to find out these commonalities and to establish criteria for comparing different approaches for value-based XML integrity constrains. Based on these criteria, some of the most prominent approaches are analyzed and finally contrasted.

Inhaltsverzeichnis

1 Einleitung **1**

1.1 Motivation 1

1.2 Zielsetzung und Lösungsansatz 2

1.3 Aufbau dieser Arbeit 2

I Grundlagen und Vergleichsmerkmale **3**

2 Grundlagen **5**

2.1 XML - eXtensible Markup Language 5

2.2 Das XML Datenmodell 6

2.3 Zugriff auf die Daten in XML Dokumenten 15

2.4 Schemasprachen 18

3 Integritätsbedingungen **23**

3.1 Einleitung 23

3.2 Integritätsbedingungen in relationalen Datenbanksystemen 24

3.3 Integritätsbedingungen in XML 25

3.4 Rahmen dieser Diplomarbeit 27

4 Vergleichsmerkmale **29**

4.1 Beschreibung der Vergleichsmerkmale 29

4.2 Zusammenfassung 83

4.3 Related Work 83

4.4 Verglichene Ansätze 88

II Vergleich von Ansätzen wertbasierter Integritätsbedingungen in XML **93**

5 Ansatz des World Wide Web Consortium **95**

5.1 Kurzbeschreibung 95

5.2 Überprüfung der Vergleichsmerkmale 96

5.3 Zusammenfassung 104

6 Ansatz von Buneman et al. **107**

6.1 Kurzbeschreibung 107

6.2 Überprüfung der Vergleichsmerkmale 107

6.3 Zusammenfassung 115

7 Ansatz von Arenas und Libkin **117**

7.1 Kurzbeschreibung 117

7.2 Überprüfung der Vergleichsmerkmale 118

7.3 Zusammenfassung 124

8 Ansatz von Fan und Libkin **127**

8.1 Kurzbeschreibung 127

8.2 Überprüfung der Vergleichsmerkmale 128

8.3 Zusammenfassung 133

9 Ansatz von Lee, Ling, Low 135

9.1 Kurzbeschreibung 135

9.2 Überprüfung der Vergleichsmerkmale 136

9.3 Zusammenfassung 140

10 Ansatz von Vincent, Liu und Liu 143

10.1 Kurzbeschreibung 143

10.2 Überprüfung der Vergleichsmerkmale 143

10.3 Zusammenfassung 153

11 Ansatz von Vincent et al. 155

11.1 Kurzbeschreibung 155

11.2 Überprüfung der Vergleichsmerkmale 155

11.3 Zusammenfassung 160

12 Ansatz von Hartmann, Link und Kirchberg 163

12.1 Kurzbeschreibung 163

12.2 Überprüfung der Vergleichsmerkmale 164

12.3 Zusammenfassung 170

13 Ansatz von Mok 173

13.1 Kurzbeschreibung 173

13.2 Überprüfung der Vergleichsmerkmale 174

13.3 Zusammenfassung 178

14 Ansatz von Karlinger **181**

14.1 Kurzbeschreibung 181

14.2 Überprüfung der Vergleichsmerkmale 181

14.3 Zusammenfassung 186

15 Zusammenfassung **189**

15.1 Resümee 189

15.2 Schlussbemerkung und Ausblick 191

Literaturverzeichnis **205**

Kapitel 1

Einleitung

In diesem Abschnitt wird eine Einführung in das Thema dieser Diplomarbeit gegeben. Im Abschnitt 1.1 wird die Motivation dieser Arbeit vorgestellt sowie die zugehörige Problemstellung. Abschnitt 1.2 beschreibt den Lösungsansatz. Im Abschnitt 1.3 werden anschließend der Aufbau und die Gliederung dieser Arbeit vorgestellt.

1.1 Motivation

Seit der Einführung von XML (eXtensible Markup Language [17]) in den späten 1990er Jahren vom World Wide Web Consortium (kurz W3C), entwickelte sie sich zum wichtigsten Datenaustauschformat im Internet [38]. Sie ermöglicht es, Daten aus verschiedenen Quellen mit unterschiedlichen Datenformaten auszutauschen. Dies wird vor allem aufgrund der flexiblen und erweiterbaren Struktur von XML ermöglicht.[17]

Mit den Regeln der Wohlgeformtheit wird eine klar definierte Syntax der XML Dokumente angegeben, wohingegen eine einheitliche Definition der Semantik von XML Daten vernachlässigt wurde. Zur Erfassung der Datensemantik in XML Dokumenten können Integritätsbedingungen (engl.: Integrity Constraints, kurz ICs) verwendet werden. Eine Vielzahl unterschiedlicher Ansätze, um Integritätsbedingungen für XML zu definieren, wurden bisher entwickelt. [86, 87]

In der Literatur sind zahlreiche Ansätze zu finden, welche sich teilweise durch Form und Aufbau unterscheiden, ergänzen oder auch aufeinander aufbauen [35, 87]. Ein Vergleich dieser Ansätze ist wegen ihrer Verschiedenartigkeit jedoch schwierig. Eine einheitliche Möglichkeit die verschiedenen Ansätze auf Grundlage der wichtigsten Vergleichsmerk-

male gegenüberzustellen ist bisher in der Literatur nicht zu finden.

1.2 Zielsetzung und Lösungsansatz

Ziel dieser Diplomarbeit ist es, eine einheitliche Vergleichsbasis für Ansätze von XML Integritätsbedingungen[1] zu schaffen. Dafür ist zunächst eine genaue Recherche der gegenwärtigen Ansätze notwendig, um anhand der gewonnenen Erkenntnisse die Gemeinsamkeiten und Unterschiede zu identifizieren. Diese werden mit Hilfe einer Liste von Vergleichsmerkmalen definiert, welche eine Zusammenfassung sämtlicher in der Literatur als wichtig erachteten Merkmale von IC-Ansätzen darstellt.

1.3 Aufbau dieser Arbeit

Im Abschnitt 2 werden grundlegende Eigenschaften von XML erläutert. Insbesondere wird dabei auf jene Bereiche eingegangen welche im Zuge des Vergleichs der IC-Ansätze von Bedeutung sind. Diese beinhalten u.a. die Verwendung eines Schemas, die Art des Zugriffs auf Daten in XML Dokumenten sowie das Datenmodell.

Der Abschnitt 3 geht anschließend auf die verschiedenen Formen von Integritätsbedingungen ein. Als erstes werden Integritätsbedingungen in relationalen Datenbanksystemen beschrieben. Anschließend werden Integritätsbedingungen in XML behandelt. Hierbei wird eine einheitliche Syntax der Constraints angegeben, welche im weiteren Verlauf dieser Arbeit verwendet wird. Des Weiteren wird kurz auf die von Karlinger [49] vorgestellte Kategorisierung von XML Integritätsbedingungen eingegangen. Abschließend wird der genaue Rahmen dieser Arbeit festgelegt.

Im Abschnitt 4 wird die Liste der Vergleichsmerkmale vorgestellt, mit Hilfe derer ein Vergleich von IC-Ansätzen ermöglicht wird. Des Weiteren wird ein Überblick zu den Arbeiten anderer Autoren gegeben, die sich mit diesem Thema beschäftigt haben.

In den Abschnitten 5 bis 14 werden anschließend verschiedene Ansätze für XML Integritätsbedingungen vorgestellt und anhand der im Abschnitt 4 vorgestellten Vergleichsmerkmale analysiert.

Abschließend wird im Abschnitt 15 eine kurze Übersicht der verglichenen Ansätze gegeben und die in dieser Arbeit gewonnenen Erkenntnisse zusammengefasst.

[1] Im Folgenden abgekürzt mit IC-Ansätze

Teil I

Grundlagen und Vergleichsmerkmale

Kapitel 2

Grundlagen

In diesem Kapitel werden grundlegende Informationen zu XML und Integritätsbedingungen gegeben. Im Abschnitt 2.1 wird XML kurz allgemein vorgestellt und anschließend im Abschnitt 2.2 das XML Datenmodell erläutert. Anschließend wird im Abschnitt 2.3 auf die Möglichkeiten des Zugriffs auf die Daten eines XML Dokuments eingegangen, gefolgt von einer kurzen Beschreibung verschiedener XML Schemasprachen (Abschnitt 2.4).

2.1 XML - eXtensible Markup Language

Aufgrund des stetigen Wachstums des Internets in den letzten Jahrzehnten wuchs auch die Nachfrage nach einem einheitlichen Format für den Datenaustausch. Dieser Datenaustausch ist jedoch wegen der heterogenen Struktur der Daten sehr schwierig und komplex. [78]

Aus diesem Grund wurde XML (Kurzform für eXtensible Markup Language [17]) in den späten 1990er Jahren von einer Arbeitsgruppe des World Wide Web Consortiums (W3C) entwickelt. XML ist flexibel, einfach in der Anwendung und dennoch ausdrucksstark genug, um den Anforderungen einer Bandbreite von Applikationen gerecht zu werden. [17]

XML ermöglicht den Austausch und die Verarbeitung von Daten aus verschiedenen Quellen, selbst wenn zu Beginn keine Informationen zur Semantik dieser Daten vorhanden sind. Darüber hinaus kann fast jedes Datenformat mit XML ausgetauscht werden. [53] XML findet auch Anwendung in den Bereichen Datenintegration, Data Warehousing, Data Translation und Data Publishing [73].

XML ist eine sogenannte Auszeichnungssprache (engl.: Markup Language), die den Dateninhalt mit Informationen über diese Daten in Form von Markup kombiniert. Dazu werden Tags verwendet. [14] Auf die Verwendung von Tags wird im nächsten Abschnitt genauer eingegangen.

XML ist jedoch keine starre, vordefinierte Auszeichnungssprache, sondern eine "Metasprache[2] für die Definition anwendungsspezifischer Auszeichnungssprachen"[46]. XML hat zudem den Vorteil, dass sie erweiterbar ist. Dies bedeutet, dass es vorher keine fixierte Menge von Tags gibt. Stattdessen ermöglicht XML eine beliebige Erweiterung bzw. Neudefinierung der Tags. [66]

2.2 Das XML Datenmodell

In diesem Abschnitt wird das XML Datenmodell vorgestellt. XML Daten werden in Form von XML Dokumenten gespeichert, welche wiederum aus einer Sequenz von Unicode-Zeichen bestehen. Der Aufbau eines XML Dokuments wird im Folgenden beschrieben.

2.2.1 XML Dokumentstruktur

Die wichtigste Anforderung an ein XML Dokument ist, dass es wohlgeformt sein muss. Dies bedeutet, dass das Dokument einer Reihe von syntaktischen Regeln, spezifiziert vom W3C, entsprechen muss. Um die Einhaltung dieser Regeln zu kontrollieren und um herauszufinden, ob ein XML Dokument wohlgeformt ist, werden XML Prozessoren, auch genannt XML Parser, verwendet. [17]

Zum einfacheren Verständnis dieser Regeln und zum Beschreiben der Dokumentstruktur wird das in Abbildung 2.1 gezeigte kurze XML Dokument als Rahmenbeispiel verwendet. Es zeigt einen Ausschnitt einer Universitätsdatenbank in Form eines XML Dokuments, das die beiden Bereiche `lvas` (Lehrveranstaltungen[3]) und `vortragende` beinhaltet. Für jede LVA wird der Titel sowie der Name des Vortragenden (`vt`) gespeichert. Der Bereich `vortragende` beinhaltet die Daten der Vortragenden, also deren Name und das Institut an dem sie arbeiten.

Jedes XML Dokument beginnt mit dem sogenannten Prolog, gefolgt von dem eigentlichen Inhalt des Dokuments. [72]

[2] Eine Metasprache wird in [46] definiert als "eine Sprache, die zur Beschreibung einer anderen Sprache verwendet wird".

[3] Im Folgenden abgekürzt mit LVA.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE library SYSTEM "uni.dtd">

<uni>
  <lvas>
    <lva titel="VO DKE">
      <vt>
        <name>
          <vn>Laura</vn>
          <nn>Leitner</nn>
        </name>
      </vt>
    </lva>
    <lva titel="UE DKE">
      <vt>
        <name>
          <vn>Felix</vn>
          <nn>Schnell</nn>
        </name>
      </vt>
    </lva>
  </lvas>
  <vortragende>
    <vt>
      <name>
        <vn>Laura</vn>
        <vn>Leitner</vn>
      </name>
      <institut>DKE</institut>
    </vt>
    <vt>
      <name>
        <vn>Felix</vn>
        <vn>Schnell</vn>
      </name>
      <institut>DKE</institut>
    </vt>
  </vortragende>
</uni>
```

Abbildung 2.1: Rahmenbeispiel

Prolog

Ein Prolog kann in eine XML Deklaration und eine optionale Dokument Type Declaration unterteilt werden. [72] Der Prolog im XML Dokument in Abbildung 2.1 sieht folgendermaßen aus:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE library SYSTEM "uni.dtd">
```

Der Prolog beginnt mit der Deklaration der XML Version mit Hilfe des Attributs `version`. Im Beispiel wurde die Version "1.0" verwendet. Die Angabe einer XML Version ist verpflichtend für jeden Prolog. Die beiden weiteren Attribute `encoding` und `standalone` sind optional. `encoding` spezifiziert die verwendete Zeichen-Deklaration der Kodierung, z.B. "UTF-8" oder "UTF-16". Mit dem Attribut `standalone` wird angegeben, ob ein Schema (siehe Abschnitt 2.4) für das XML Dokument verwendet wird und falls ja, ob dieses Schema innerhalb des Dokuments eingebunden ist oder als externe Datei vorhanden ist. Im Rahmenbeispiel wird eine Schemadatei namens "uni.dtd" referenziert. Wäre das Schema direkt im XML Dokument angefügt, müsste es gleich nach dem Prolog beginnen. [17]

Eigentlicher Inhalt

Nach dem Prolog kommt der eigentliche Inhalt des XML Dokuments welcher in Form von Elementen, Attributen, Textwerten sowie möglichen Verarbeitungsanweisungen (engl.: Processing Instructions) oder Kommentaren angegeben wird. [66]

Die Elemente eines XML Dokuments werden mit Hilfe von Tags ausgedrückt. Jedes Element hat einen Start-Tag <`elementname`>, der den Namen (auch genannt Label) eines Elements enthält, und einen zugehörigen End-Tag </`elementname`>. Elemente können Unterelemente sowie Text beinhalten oder leer sein. Die Elemente eines XML Dokuments müssen richtig verschachtelt sein, d.h., die Start- und End-Tags der Elemente dürfen sich nicht überschneiden. Am Anfang eines XML Dokuments steht das Wurzelelement, das den Namen des Dokuments beinhaltet. [72]

Des Weiteren kann ein Element auch Textwerte in Form von Zeichenfolgen beinhalten. Im Gegensatz zu Elementen, die jeweils ein eigenes Label haben, werden bei Textwerten keine eigenen Labels verwendet. [53]

Neben Unterelementen und Textwerten können in einem Element auch Attribute vorhan-

den sein. Jedes Attribut hat ein Label und einen Wert, spezifiziert in Form von <element attributname = "Wert"/>. Die Attribut-Werte müssen dabei unter (einfache oder doppelte) Anführungszeichen gesetzt werden. [17]

Innerhalb eines Elements ist es nicht erlaubt mehrere Attribute mit demselben Label zu verwenden. Diese Regelung weicht erheblich von den Bestimmungen der Unterelemente innerhalb eines Elements ab, wo mehrfaches Vorkommen gleichnamiger Elemente erlaubt ist. Es ist jedoch möglich, dass in einem Element mehrere Attribute mit unterschiedlichen Labels vorkommen oder dass Attribute mit demselben Namen innerhalb verschiedener Elemente auftauchen. [72]

Beispiel 2.1: *Das XML Dokument in Abbildung 2.1 beinhaltet u.a. die Elemente* `lvas`, `lva`, `vt` *und* `name`, *welche korrekt verschachtelt sind. Das Element* `vn` *der ersten Vortragenden beinhaltet den Wert "Laura". Des Weiteren beinhaltet jedes Element* `lva` *ein Attribut mit dem Label* `titel` *sowie einen dazugehörigen Wert.*

Processing instructions (kurz PIs) sind optional und können verwendet werden, um den Applikationen, die das XML Dokument verarbeiten, spezielle Anweisungen zu übermitteln. Ebenso können Kommentare in ein XML Dokument integriert sein. Sowohl PIs als auch Kommentare werden vom Parsing-Prozess, also dem Prozess um XML Dokument auf ihre Wohlgeformtheit zu überprüfen, nicht berücksichtigt. [72] Da sie weder die Syntax noch die Semantik von XML Daten beeinflussen, werden sie im weiteren Verlauf dieser Diplomarbeit nicht weiter berücksichtigt.

Weitere Eigenschaften von XML Dokumenten

Die soeben beschriebenen Regeln des W3C wurden von verschiedenen Datenmodellen erweitert. Die weit verbreitetsten Datenmodelle sind das XML Information Set [26], das Document Object Model (DOM) [56] und das in XPath [12] und XQuery [16] verwendete Baummodell. Im Zusammenhang mit diesen Baummodellen ist zu beachten, dass deren Erweiterungen nicht als Voraussetzung für wohlgeformte XML Dokumente gemäß der XML Spezifikation [17] gelten.

Eine wichtige Erweiterung betrifft die Dokumentordnung. Sie wird vom W3C im XPath und XQuery Datenmodell [16] definiert als jene Reihenfolge, in der die Knoten in einem XML Dokument vorkommen. Die Hierarchie der Knoten (siehe Abschnitt 2.2.4) muss dabei eingehalten werden. Bezüglich der Attribute ist hierbei zu beachten, dass diese innerhalb eines Elements nicht geordnet sein müssen, sie jedoch direkt nach dem Element

im XML Dokument, dem sie zugeordnet sind, vorkommen müssen. [16]

Wie im weiteren Verlauf dieser Arbeit ersichtlich wird, gibt es auch Baummodelle, in denen diese Eigenschaft nicht übernommen wird. Hierauf wird im Abschnitt 4.1.3 näher eingegangen.

Eine weitere wichtige Eigenschaft von XML Dokumenten betrifft die Textwerte eines Elements. Es ist möglich, dass ein Element sowohl Textwerte als auch Unterelemente beinhaltet. Dadurch wird gemischter Inhalt (engl.: mixed content) erlaubt. [53]. Im Folgenden wird ein Beispiel gezeigt, das diese Eigenschaft verdeutlicht.

Beispiel 2.2: *Ein Buch besteht aus mehreren Kapiteln und Unterkapiteln, welche den eigentlichen Text des Buches beinhalten. Zwischen bzw. vor den einzelnen Kapiteln werden Einleitungen und Überleitungen eingefügt. Wird dieses Buch nun in Form eines XML Dokuments gespeichert, werden mehrere Elemente* `kapitel` *benötigt, die wiederum* `kapitel`*-Unterelemente beinhalten können. Ebenfalls werden in bzw. zwischen den einzelnen* `kapitel`*-Elementen Textwerte gespeichert, welche den eigentlichen Inhalt des Buches wiedergeben. Ein Element* `kapitel` *kann somit sowohl Unterelemente als auch Textwerte beinhalten. Für eine korrekte Umsetzung ist hierbei die Reihenfolge der Unterelemente und Textwerte einzuhalten.*

Auf das Vorhandensein von gemischtem Inhalt in XML Dokumenten wird ebenfalls im Abschnitt 4.1.3 näher eingegangen.

2.2.2 Semistrukturierte Daten

XML Daten werden meist aus anderen Datenbanken, wie z.B. relationalen oder objektorientierten, generiert. Relationale Daten haben eine fixe, vordefinierte Datenstruktur. Sie bestehen aus einer beliebigen Anzahl von Tabellen, die wiederum Spalten und Reihen beinhalten. Die Namen der Spalten bezeichnen die Attribute der Relationen, während in den einzelnen Reihen (auch genannt Tupel oder Entitäten) die Datenwerte gespeichert werden. Die erlaubten Datentypen (z.B. string, integer, boolean, ...) werden im Vorhinein spezifiziert. Jede Entität einer Relation wird mit Hilfe eines Schlüsselattributs eindeutig identifiziert. [3]

Objektorientierte Daten sind ebenfalls strukturiert. [77] Im Gegensatz zu relationalen Datenbanken werden Daten nicht in Form von Tupel, sondern als Objekte gespeichert. Diese werden durch einen eindeutigen *object identifier* (kurz: oid) identifiziert. [30]

Im Gegensatz zu strukturierten Daten haben unstrukturierte Daten keine formal definierte Datenstruktur. Beispiele für unstrukturierte Daten sind Grafiken oder digitale Tonaufnahmen. [10]

Häufig können Daten nicht eindeutig den strukturierten und unstrukturierten Daten zugeordnet werden. Diese Art von Daten werden als semistrukturiert bezeichnet. [1] Ein typisches Beispiel dafür sind die Daten einer E-Mail. Während der Absender und der Empfänger als strukturierte Daten dargestellt werden können, ist der eigentliche Text der E-Mail unstrukturiert.

Semistrukturierte Daten haben eine irreguläre und flexible Datenstruktur. Dies bedeutet, dass sie nicht anhand eines einheitlichen, vordefinierten Schemas aufgebaut werden müssen. Es ist somit nicht zwingend erforderlich, dass eine Schemaspezifikation vorhanden ist, sodass semistrukturierte Daten auch unabhängig von einem Schema verwendet werden können. Vielmehr sind semistrukturierte Daten selbstbeschreibend, d.h., sie definieren ihre eigenen Struktur unter Zuhilfenahme von Tags. Aufgrund ihrer flexiblen Datenstruktur ist es auch möglich, dass einzelne Datenobjekte fehlen, oder, im gegenteiligen Fall, sie mehrfach vorhanden sind[4]. [2] Hierauf wird im Abschnitt 4.1.7 näher eingegangen.

XML Dokumente sind semistrukturiert. [78]

Semistrukturierte Daten können in Form eines gerichteten Graphen dargestellt werden, dessen Knoten mittels Labels beschriftet werden [21]. Gemäß Diestel [29] wird ein Graph folgendermaßen definiert (informell):

> *Ein Graph ist ein Paar (V, E), wobei V eine endliche Menge von Knoten bezeichnet und E eine Menge von Kanten, die zwei Knoten miteinander verbinden. Eine Kante ist also ein Paar* (v_1, v_2)*, mit* $v_1, v_2 \in V$*. Ein Graph wird als gerichteter Graph bezeichnet, wenn dessen Kanten immer von einem Anfangsknoten zu einem Zielkonten führen.* [29]

2.2.3 Datenorientiertes vs. dokumentorientiertes XML

XML Dokumente können je nach Anwendungsbereich in datenorientierte und dokumentorientierte unterteilt werden. [66]

[4] Im weiteren Verlauf dieser Diplomarbeit wird der Begriff "Datenobjekt" im Sinne der Definition in [46] verwendet: Ein Datenobjekt bezeichnet *"die Zusammenfassung von digitalen Daten zu einer Einheit [...]"*. Sofern nichts anderes angegeben wird, steht diese Bezeichnung im Folgenden für ein Element, ein Attribut oder einen Textwert.

Datenorientierte XML Dokumente weisen eine sehr regelmäßige Struktur auf und basieren oft auf einem vordefinierten Schema. Die Elemente beinhalten meist nur Unterelemente oder Textwerte, selten jedoch beides. Die Reihenfolge der Geschwisterelemente wird meist als unwichtig erachtet und die Elementhierarchie ist in der Regel eher flach. [66]

Datenorientierte XML Dokumente werden vordergründig zum Datenaustausch verwendet und sollen daher für Maschinen effizient lesbar sein. Sie haben große Ähnlichkeit mit traditionellen Datenbanksystemen wie beispielsweise relationale Datenbanken. Beispiele für datenorientierte XML Dokumente sind Personendaten, Kundenbestellungen, wissenschaftliche Daten, Finanzdaten usw. [66]

Im Vergleich dazu haben dokumentorientierte XML Dokumente eine eher unregelmäßige Struktur. Die Elemente können sowohl Unterelemente als auch Textwerte enthalten (gemischter Inhalt) und die Reihenfolge der Geschwisterelemente ist meist von Bedeutung. [66]

Dokumentorientierte XML Dokumente sollen auch für Menschen leicht lesbar sein und werden beispielsweise zur Speicherung bzw. Übermittlung von Emails, Büchern oder anderen Publikationen verwendet. [66]

Oft ist eine genaue Einordnung in daten- oder dokumentorientierte XML Dokumente nicht möglich. Beispielsweise kann eine Kundenrechnung sowohl reguläre Daten (Name, Adresse, Produkte) als auch zusätzliche Informationen in Form von einfachen Text (z.B. eine detaillierte Beschreibung eines Produkts) enthalten. Ein anderes Beispiel hierfür sind diverse Webseiten, die einerseits einer einheitlichen Struktur unterliegen, andererseits aber auch viel Freiraum für unstrukturierte Daten bereitstellen sollen. [54]

2.2.4 XML Baum

Da XML Daten semistrukturierte Daten darstellen, sind sie irregulär, selbstbeschreibend und können unabhängig von einem vordefinierten Schema verwendet werden. Aufgrund ihrer Wohlgeformtheit haben XML Daten eine hierarchische Struktur, die in der Form eines Baumes dargestellt werden kann. [21] Ein Baum ist ein gerichteter Graph, der jedoch keine Zyklen enthalten darf. Des Weiteren existiert genau ein Knoten *root*, von dem aus jeder weitere Knoten im Baum über genau einen Pfad erreichbar ist. Ein Pfad ist eine nichtleere Folge von Knoten in einem Baum, die von einem Ausgangsknoten zu einem Zielknoten führt. Knoten, die keine ausgehenden Kanten haben, werden Blattknoten genannt. [29]

In einem XML Baum gibt es drei verschiedene Arten von Knoten: Element-, Attribut- und Textknoten. Elementknoten repräsentieren die Elemente eines XML Dokuments, welche durch ihre Start- und End-Tags spezifiziert werden. Das Wurzelelement eines XML Dokuments wird auch in einem XML Baum als Wurzelknoten bezeichnet. Unterhalb eines Elementknotens können weitere Elementknoten vorhanden sein, ebenso wie Attribut- und Textknoten. Manche Baummodelle fordern, dass die Elementknoten gemäß ihres Vorkommens im XML Dokument geordnet sein müssen (siehe Abschnitt 2.2.1). Ein Element x, dass vor einem Element y im XML Dokument auftritt, wird als Elementknoten x links vom Elementknoten y im Baum positioniert. Attribut- und Textknoten müssen Blattknoten sein. Element- und Attributknoten besitzen ein Label und einen Wert (gemäß des darzustellenden XML Dokuments). Textknoten besitzen einen Wert, haben aber kein zugehöriges Label. [53]

Abbildung 2.2 zeigt die Baumdarstellung des XML Dokuments von Abbildung 2.1 und wird im folgenden Beispiel erklärt:

Beispiel 2.3: *Der Wurzelknoten hat das Label* `uni`*. Beispiele für Elementknoten sind* `uni`*,* `lvas`*,* `lva`*,* `vt` *und* `name`*. Attributknoten sind die Knoten mit dem Label* `titel` *mit ihren zugehörigen Werten* `"VO DKE"` *und* `"UE DKE"`*. Da Textknoten kein Label haben, werden sie direkt mit dem jeweiligen Textwert repräsentiert. Beispiele für Textwerte sind* `"Laura"`*,* `"Leitner"`*,* `"Felix"` *und* `"Schnell"`*.*

Die Darstellung von XML Dokumenten als XML Bäume zeigt bereits, dass die einzelnen Kanten die Hierarchiestufen der XML Daten repräsentieren. Die Beziehungen der Knoten zueinander sind von großer Bedeutung, vor allem um einen einfachen Zugriff auf die Daten zu ermöglichen (mehr dazu siehe Abschnitt 2.3). [66]

Ein Knoten *y* der im Baum eine Ebene unter einem anderen Knoten *x* auftaucht, wird als Kind-Knoten von Knoten *x* bezeichnet. Knoten *y* ist also ein direkt untergeordneter Knoten von *x*. Die Achse[5] zwischen Knoten *x* und Knoten *y* wird als Kind-Achse bezeichnet. Die Kind-Knoten von Knoten *x* sowie sämtliche Kindes-Kind-Knoten usw. werden Nachkommen von Knoten *x* genannt. [66]

Kanten können auch die Beziehungen zu Knoten höherer Ebenen darstellen. Folglich wird ein Knoten *x* der eine Hierarchiestufe über dem Knoten *y* steht als Eltern-Knoten von *y* bezeichnet. Sämtliche Knoten die über dem Knoten *y* stehen werden Vorfahren-Knoten genannt. Jeder Knoten hat genau einen Eltern-Knoten, ausgenommen dem Wurzel-Knoten

[5] Eine Achse beschreibt die Beziehung zwischen zwei Knoten. [12]

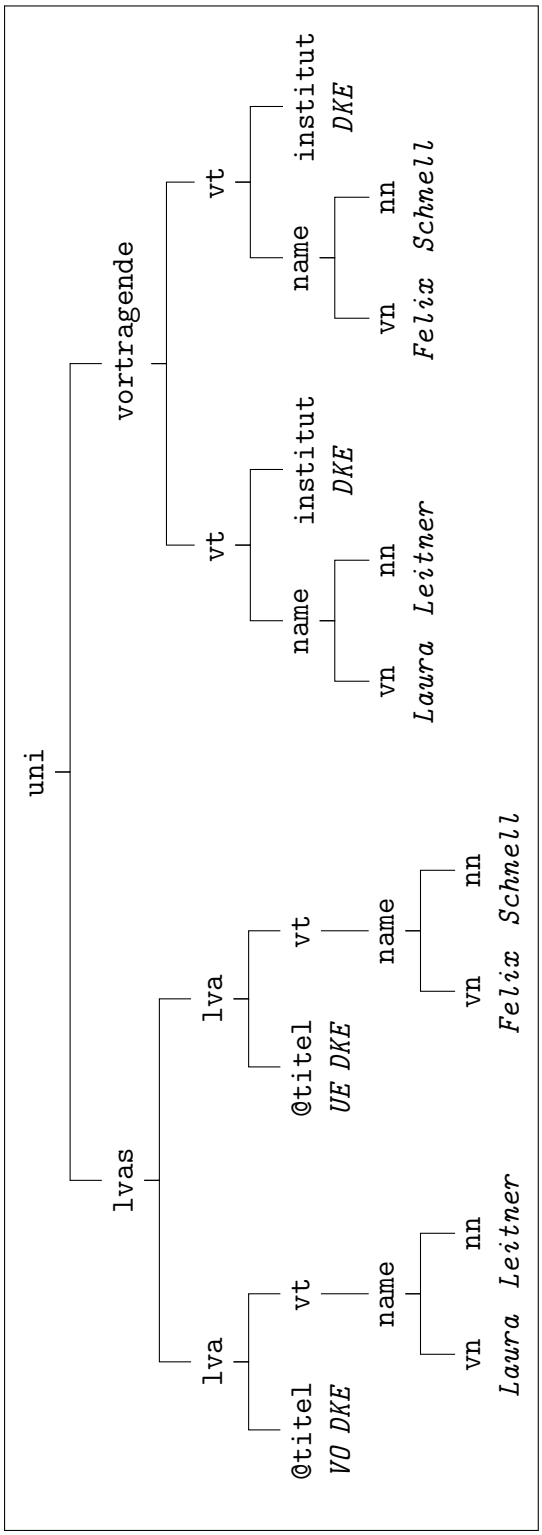

Abbildung 2.2: Baumdarstellung des Rahmenbeispiels

der auf der höchsten Ebene des Baumes steht und somit keine Vorfahren hat. [66]

Darüber hinaus ist es auch möglich Seitwärts-Achsen zu definieren. Diese Achsen werden Geschwister-Achsen genannt und bezeichnen die Beziehung von zwei Knoten *x* und *y* die denselben Eltern-Knoten haben. Dabei wird zwischen Nachfolgende-Geschwister und Vorangehende-Geschwister unterschieden, abhängig von ihrer Position im XML Baum bzw. der Anordnung der Elemente im XML Dokument. [66]

Im folgenden Beispiel werden die verschiedenen Achsen anhand des XML Baums in Abbildung 2.3 nochmals verdeutlicht.

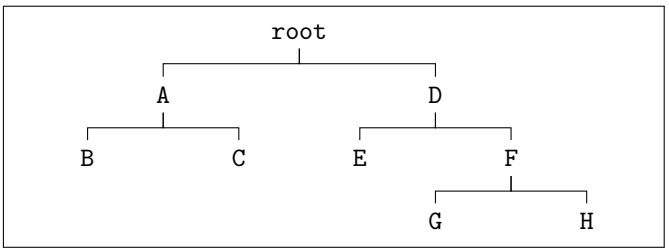

Abbildung 2.3: XML Baum zur Darstellung der Achsen

Beispiel 2.4: *Der XML Baum in Abbildung 2.3 ist aufgeteilt in vier Ebenen. Der oberste Knoten ist der* `root` *Knoten, gefolgt von den weiteren Knoten* `A - H`*. Der* `root` *Knoten ist der Eltern-Knoten von den beiden Knoten* `A` *und* `D`*, die wiederum die Eltern-Knoten der Knoten* `B` *und* `C` *bzw.* `E` *und* `F` *sind. Des Weiteren ist der* `root` *Knoten der Vorfahren-Knoten aller anderen Knoten im Baum, d.h., der Knoten* `A, B, C, D, E, F, G` *und* `H`*. In der entgegengesetzten Sichtweise sind beispielsweise die Knoten* `B` *und* `C` *Kind-Knoten von* `A`*, und Nachkommen-Knoten vom Knoten* `root`*. Darüber hinaus sind die Knoten* `A` *und* `D` *Geschwister-Knoten, wobei* `A` *ein Vorangehende-Geschwister-Knoten von* `D` *ist.*

2.3 Zugriff auf die Daten in XML Dokumenten

Um auf die Daten in einem XML Dokument zugreifen zu können, werden zuerst die jeweiligen Datenobjekte adressiert. Dazu werden die eben beschriebenen Achsen verwendet. [18] Um einen bestimmten Knoten auszuwählen, wird ein Pfad von einem Ausgangsknoten zu dem zu selektierenden Knoten definiert. [42]

Die grundlegende Form von Pfaden ist der Abwärts-Pfad, also ein Pfad der von einem Knoten *x* die einzelnen Ebenen hinunter navigiert bis zum Knoten *y*. Die Kind- bzw.

Nachkommen-Achsen werden dabei in Form von einfachen Verkettungsoperatoren repräsentiert. Aufgrund der hierarchischen Datenstruktur von XML gibt es immer genau einen Pfad, der vom Wurzelknoten zu jedem einzelnen Knoten im Baum führt. Buneman et al. [18] definieren einen Pfad beispielsweise folgendermaßen:

> *Ein Pfad p wird definiert als p =* $l_0.\cdots.l_n$*, mit* l_i*,* $i \in \{0, \cdots, n\}$*, als Element-, Attribut- oder Textlabel*[6] *,* l_n *ist das Label des zu adressierenden Knotens und das Symbol "." wird als Verkettungsoperator verwendet. Falls kein Label im Pfad vorhanden ist, wird von einem leeren Pfad gesprochen (Symbol "*ϵ*").* [18]

Hierbei wird bereits ersichtlich, dass in XML die Labels der Knoten zum Definieren der Pfade verwendet wird.

Zur Unterscheidung zwischen Element- und Attribut-Labels in einem Pfad wird bei einem Attribut-Label oft das Symbol "@" vor dem eigentlich Label angegeben. [66] Im folgenden Beispiel sind einige Pfade für den XML Baum in Abbildung 2.2 aufgelistet, wobei als Verkettungsoperator das in [18] angegebene Symbol "." verwendet wird.

Beispiel 2.5: *In der nachfolgenden Tabelle werden einige Pfade vom XML Baum in Abbildung 2.2 angegeben und beschrieben. Hierbei ist zu beachten, dass* `S` *als Label für Textknoten verwendet wird.*

`uni.lvas`	*Pfad zum Elementknoten mit dem Label* `lvas`
uni.lvas.lva	*Pfad zu den Elementknoten mit dem Label* `lva`
uni.lvas.lva.@titel	*Pfad zu den Attributknoten mit dem Label* `@titel`
uni.vortragende.vt.name.vn.S	*Pfad zu den Textknoten der Elementknoten mit dem Label* `vn`
ϵ	*Der leere Pfad*

Pfade, die den Wurzelknoten als Ausgangsknoten verwenden, werden absolute Pfade genannt. Es ist aber auch möglich, Pfade zu definieren, die an einem anderen Knoten als dem Wurzelknoten beginnen. l_0 ist also nicht der Wurzelknoten des XML Baumes, sondern ein beliebig definierter Ausgangsknoten. Somit wird der Abfrageprozess auf einen

[6] Hierbei ist zu beachten, dass Attribut- und Textlabel nur an der letzten Position des Pfades vorkommen dürfen.

genau spezifizierten Teilbereich, einem sogenannten Unterbaum, beschränkt. Diese Pfade werden als relative Pfade bezeichnet. [18]

Des Weiteren gibt es die Möglichkeit, zur Definition von Pfaden einen Kleene Operator zu verwenden. Der Kleene Operator (auch genannt Kleene Closure, meist repräsentiert durch das Symbol "_*") steht für eine nicht weiter zu definierende, beliebig lange Sequenz von Knoten in einem Pfad. Soll nur ein einzelner, undefinierter Knoten in einem Pfad repräsentiert werden, kann das Wildcard-Symbol "_" verwendet werden. [41] Bsp 2.6 zeigt einige Pfade, in denen ein Kleene Closure bzw. das Wildcard-Symbol verwendet wurde.

Beispiel 2.6: *Für den XML Baum in Abbildung 2.2 könnten unter anderem folgende Pfade definiert werden:*

`_.lvas`	*Pfad zum Elementknoten mit dem Label* `lvas`
`_*.lva.@titel`	*Pfad zu den* `@titel`*-Attributknoten der Elementknoten mit dem Label* `lva`
`_*.name.vn`	*Pfad zu den* `vn`*-Elementknoten der Elementknoten mit dem Label* `name`

Neben den eben vorgestellten Abwärts-Pfaden können auch Aufwärts- und Seitwärts-Pfade definiert werden, angelehnt an die Aufwärts- und Seitwärts-Achsen in einem XML Baum. [12]

Pfade können mit Hilfe einer großen Auswahl von Pfadsprachen definiert werden. Neben sehr einfachen Notationen mit denen nur elementare Abwärtspfade definiert werden können, gibt es auch viel weiter entwickelte und teilweise auch sehr komplexe Sprachen. [42] Die wahrscheinlich populärste Pfadsprache ist XPath [12], die oft gemeinsam mit der Abfragesprache XQuery [16] verwendet wird. Diese werden im Folgenden kurz beschrieben.

XPath

XPath [12] wurde vom W3C entwickelt und wird von vielen Anwendern zum Adressieren von Knoten in XML Bäumen verwendet. Darüber hinaus wurde XPath auch in einige andere XML Standards integriert, wie beispielsweise XML Schema [15, 32, 76], XPointer [37] oder XLink [28].

XPath ermöglicht das Navigieren in sämtliche Richtungen und stellt somit Mechanismen für Abwärts, Aufwärts- und Seitwärtspfade zur Verfügung. Elementknoten können als

Kinder wiederum Elementknoten und Textknoten haben. Attributknoten werden hingegen nicht als Kinder der Elementknoten behandelt, sondern als deren "Eigenschaften". Werden nun die Kind-Knoten eines Elementknotens abgefragt, werden die Attributknoten nicht aufgelistet. Um auf Attribut-Werte zuzugreifen, müssen diese mit Hilfe des "@"-Symbols angesprochen und anschließend deren Werte abgefragt werden. [12]

Neben den Notationen zum Navigieren in XML Bäumen bietet XPath auch eine große Anzahl an Funktionen, die auf die Knoten angewendet werden können. Beispiele dafür sind Verkettungen, Filterregeln oder Zählfunktionen. Darüber hinaus können auch Programmiersprachen-ähnliche Ausdrücke mit XPath formuliert werden, wie beispielsweise *"for-in-return"* oder *"if-then-else"* Methoden. [12]

Die große Anzahl an Mechanismen und Werkzeugen macht XPath leider auch sehr komplex und somit aufwändig in der Umsetzung. Aus diesem Grund beschränken sich viele Anwender auf Fragmente von XPath. Ein Beispiel dafür ist die Schemasprache XML Schema, bei der zur Definition von Integritätsbedingungen (siehe Abschnitt 5) nur Abwärts-Pfade verwendet werden, Aufwärts- und Seitwärts-Pfade jedoch nicht. [11]

XQuery

XQuery [16] wurde ebenfalls vom W3C entwickelt. Es verwendet XPath-Ausdrücke [12] zum Adressieren von Knoten, um diese anschließend anhand vorgegebener Konditionen zu überprüfen. Dafür werden sogenannte FLWOR-Ausdrücke verwendet. FLWOR steht für *for, let, where, order by* und *return* und ermöglicht somit das Formulieren von Abfragen die ähnlich wie beispielsweise SQL-Abfragen in relationalen Datenbanken aufgebaut sind. [16]

2.4 Schemasprachen

Wie bereits im Abschnitt 2.2.2 beschrieben wurde, sind XML Daten semistrukturiert. Es ist somit nicht zwingend erforderlich, dass eine Schemaspezifikation vorhanden ist. Dennoch werden oft Schemata verwendet. [66] Ein großer Vorteil von Schemaspezifikationen ist, dass durch die definierte Dokumentstruktur der Zugriff auf die Daten vereinfacht wird und dadurch Datenabfrage, Datenintegration sowie Update-Prozesse verbessert werden können. [71]

Um zu prüfen, ob ein XML Dokument einem Schema entspricht und somit alle definierten

Anforderungen erfüllt, wird ein Schema-Prozessor, auch genannt XML Parser, verwendet. Besteht ein XML Dokument diese Prüfung, wird es als "gültig" (engl.: valid) bezeichnet. [17] Ein gültiges XML Dokument ist von einem wohlgeformten zu unterscheiden. Wohlgeformt bedeutet, dass die allgemeinen Regeln der XML Spezifikation eingehalten werden, während ein gültiges XML Dokument gegenüber einem Schema validiert wird. Somit muss ein wohlgeformtes Dokument nicht unbedingt gültig sein, wohingegen jedes gültige Dokument auch wohlgeformt sein muss. [72]

Neben der Spezifizierung der Dokumentstruktur kann auch die Datensemantik mit Hilfe von XML Schemata definiert werden. Dies erfolgt in Form von Integritätsbedingungen [66], welche im weiteren Verlauf dieser Arbeit noch genauer beschrieben werden (siehe Kapitel 3). Zuerst wird jedoch etwas näher auf XML Schemata im Allgemeinen, sowie einige wichtige und oft verwendete Schemasprachen eingegangen.

Mittlerweile gibt es eine große Anzahl von Schemasprachen, die sich teilweise sehr stark in ihren Fähigkeiten unterscheiden. Eine Schemasprache soll einfach genug sein, um eine effiziente Anwendung zu ermöglichen, gleichzeitig aber auch ausdrucksstark genug, um den Anforderungen der Anwender zu entsprechen. [66]

Im folgenden Abschnitt werden die beiden bekanntesten Schemasprachen DTD (Document Type Definition) [17] und XML Schema [15, 32, 76] beschrieben. Darüber hinaus wurden noch weitere Schemasprachen entwickelt, wie beispielsweise RelaxNG [25], DSD 2 [65], Schematron [48, 63] oder SOX [27]. Eine genauere Analyse unterschiedlicher Schemasprachen kann u.a. in [2, 52] und [57] gefunden werden.

2.4.1 DTD - Document Type Definition

DTD, die Document Type Definition [17], ist die erste vom W3C entwickelte Schemasprache. Sie ist eine sehr einfache Schemasprache, leicht zu erlernen, jedoch mit eingeschränkten Möglichkeiten. DTDs werden nicht in der XML Notation geschrieben, was beispielsweise dazu führen kann, dass zusätzliche Editoren benötigt werden. Dennoch sind DTDs weit verbreitet und sie ist vermutlich die meist verwendete Schemasprache für XML. [66]

DTDs werden verwendet, um Grammatiken für XML Dokumente zu definieren. Alle XML Dokumente, die den Grammatiken einer bestimmten DTD entsprechen, sind vom gleichen Dokumenttyp. Oft werden sie auch als Dokumentinstanz der jeweiligen DTDs bezeichnet. [66] Im Folgenden wird kurz auf die wichtigsten Funktionen von DTDs einge-

gangen. Für detailliertere Informationen wird auf die XML Spezifikation [17] verwiesen.

Elemente werden in Form von <!ELEMENT name> deklariert und können Unterelemente enthalten oder leer sein (Schlüsselwort EMPTY). Ist ein Element nicht leer, können entweder die Unterelemente aufgelistet werden oder das Schlüsselwort ANY verwendet werden, welches besagt, dass beliebige, nicht genau definierte Unterelemente erlaubt sind. Die erlaubten Wiederholungen eines Elements werden mit den Operatoren "?", "*" und "+" definiert, welche für "0 oder 1", "0 oder mehr" und "1 oder mehr" stehen. Wenn keiner dieser Operatoren angegeben wird, muss das Element genau einmal vorhanden sein. [17]

Um festzulegen, dass ein Element Zeichendaten beinhalten kann, wird die Notation PCDATA (Parsed Character Data) verwendet. Sollten Zeichendaten nicht in den Parsingprozess integriert werden, wird die Notation CDATA (Unparsed Character Data) verwendet. Dies ist vor allem dann nützlich, wenn der Text aus vielen Symbolen besteht, die in XML eine spezielle Bedeutung haben, wie beispielsweise die Symbole "<" oder ">". Es ist auch möglich, stattdessen eine sogenannte *Entitiy* für das jeweilige Symbol zu verwenden. Ein Beispiel dafür ist die *Entity* "<"welche für das Symbol "<" steht. Darüber hinaus können sie als Abkürzung für oft verwendeten Text definiert werden (z.B. die *Entity* "&jku" für den Namen "Johannes Kepler Universität") oder auch zum Spezifizieren von Links zu externen Datafiles verwendet werden. [17]

Die Attribute eines Elements werden mit dem Operator <!ATTLIST> angeführt. Attribute können einen bestimmten Typ haben, beispielsweise CDATA. Die Auswahl der Attributtypen in DTDs ist jedoch sehr limitiert, sodass beispielsweise die Typen integer oder URI nicht integriert sind. Es ist nicht möglich benutzerdefinierte Datentypen in DTDs zu verwenden. [66]

Zwei sehr wichtige Attributtypen in DTDs sind ID und IDREF. ID-Attribute werden verwendet um Elemente im XML Dokument eindeutig zu identifizieren. IDREF hingegen referenziert auf ein bereits existierendes ID-Attribut eines anderen Elements. ID und IDREF-Attribute sind mit Schlüsseln und Fremdschlüsseln in relationalen Datenbanken zu vergleichen (siehe Abschnitt 3.2) – jedoch mit einigen Einschränkungen. So müssen ID-Attribute im gesamten XML Dokument eindeutig sein und können nicht auf einzelne Bereiche beschränkt werden, wie dies beispielsweise für einzelne Relationen in relationalen Datenbanken möglich ist. Des Weiteren können nur unäre Schlüssel zum Identifizieren verwendet werden, also Schlüssel die aus genau einem Attribut bestehen. [33].

2.4.2 XML Schema

XML Schema [15, 32, 76] ist die zweite vom W3C veröffentlichte Schemasprache für XML. Sie wurde entwickelt, um die Einschränkungen von DTD, vor allem bezüglich ihrer Ausdrucksstärke, zu überwinden. Ein weiterer Vorteil von XML Schema ist, dass es selbst auch in der XML Syntax geschrieben wird. [66]

In XML Schema werden Elemente definiert mit `<element name="name" type="type"/>` und Attribute mit `<attribute name="name" type="type"/>`. Als Typen können einfache oder komplexe Typen verwendet werden, wobei Attribute auf einfache Typen beschränkt sind. Einfache Typen erlauben nur textbasierten Inhalt in Form von primitiven Datentypen (z.B. string, boolean, date, usw.) oder davon abgeleiteten Datentypen (z.B. language, integer, nonPositiveInteger, usw.). Mit einem komplexen Typ werden die Kinder bzw. Eigenschaften eines Elements definiert, also seine Unterelemente, Attribute sowie Character Data der Elemente. Des Weiteren unterstützt XML Schema auch die Verwendung von benutzerdefinierten Datentypen. [66]

Darüber hinaus erlaubt XML Schema auch die Vererbung von Attributen und Elementen. Ebenso wie in DTDs können auch in XML Schema *Entities* verwendet werden. [57]

Ein sehr nützlicher Mechanismus von XML Schema sind Namensräume. Sie werden verwendet, um die Eindeutigkeit von Element- und Attributnamen zu gewährleisten. Elemente und Attribute, die zwar den gleichen Namen haben, jedoch mit verschiedener Bedeutung, werden unterschiedlichen Namensräumen zugeordnet. Dadurch können Namenskonflikte vermieden werden, welche beispielsweise beim Zusammenfügen und Integrieren verschiedener XML Dokumente entstehen können. [57]

Darüber hinaus bietet XML Schema auch Operatoren zum Definieren der Kardinalität von Elementen und Attributen, wie beispielsweise `<minOccurs>` und `<maxOccurs>`. Um zu definieren, dass ein Element oder Attribut eindeutig sein soll, kann die Notation `<unique>` verwendet werden. [57] Im Gegensatz zu DTD unterstützt XML Schema auch ungeordnete Elemente, welche mit dem `"&"`-Operator verbunden werden. [47]

Des Weiteren ist es möglich, Schlüssel und Fremdschlüssel mit Hilfe der Typen `key` bzw. `keyref` zu definieren. [76] Hierauf wird detailliert im Kapitel 5 eingegangen.

Die bisher gezeigten Mechanismen sind nur eine kleine Auswahl der vielen Möglichkeiten von XML Schema. Weitere Funktionen können in der XML Schema Spezifikation [15, 32, 76] nachgelesen werden.

Die vielen Vorzüge von XML Schema machen die Sprache leider auch sehr kompliziert, was sie vor allem für Nicht-Experten schwer anwendbar macht. Anwender beschränken sich deshalb oft auf die Grundfunktionen und verzichten auf die komplexeren Features. [66]

Kapitel 3

Integritätsbedingungen

Dieses Kapitel beschäftigt sich mit den verschiedenen Formen und Kategorien von Integritätsbedingungen. Nach einer kurzen Einleitung werden im Abschnitt 3.2 Integritätsbedingungen in relationalen Datenbanksystemen vorgestellt. Aufbauend darauf wird im Abschnitt 3.3 auf Integritätsbedingungen in XML eingegangen und die verschiedenen Kategorien von XML Integritätsbedingungen werden erläutert. Abschließend wird im Abschnitt 3.4 der Rahmen dieser Diplomarbeit festgelegt.

3.1 Einleitung

Bisher wurde hauptsächlich die Datenstruktur bzw. die Syntax von XML Dokumenten beschrieben. Nun wird etwas näher auf die Datensemantik von XML Dokumenten eingegangen, also die Bedeutung der Daten. Diese kann mit Hilfe von Integritätsbedingungen (engl.: Integrity Constraints, im Folgenden abgekürzt mit ICs) definiert werden. [3]

Integritätsbedingungen sind vor allem aus traditionellen Datenbanksystemen bekannt, wie beispielsweise den relationalen oder objektorientierten Datenbanksystemen. Sie können – wie bereits erwähnt – zur Spezifikation der Datensemantik verwendet werden. Des Weiteren können sie das Schemadesign verbessern. Sie werden auch verwendet, um Update-Anomalien zu vermeiden und die Datenintegrität zu gewährleisten. Dazu wird geprüft, ob die Datenbank nach einem Update-Prozess noch den angegebenen Integritätsbedingungen entspricht. Integritätsbedingungen unterstützen auch eine effiziente Datenspeicherung und optimieren dadurch die Datenabfragen. [3]

3.2 Integritätsbedingungen in relationalen Datenbanksystemen

In diesem Abschnitt wird auf verschiedene Formen von Integritätsbedingungen (gemäß Abiteboul et al. [3]) eingegangen.

Die meist verwendeten Formen von relationalen Integritätsbedingungen sind Funktionale Abhängigkeiten (engl.: Functional Dependencies), Schlüssel (engl.: Keys), Inklusionsabhängigkeiten (engl.: Inclusion Dependencies) und Fremdschlüssel (engl.: Foreign Keys)[7] [3]. Selbiges gilt für Integritätsbedingungen in XML [9, 73].

3.2.1 Funktionale Abhängigkeiten

Funktionale Abhängigkeiten haben die Form $\sigma = \{X\} \rightarrow \{Y\}$, wobei {X} und {Y} jeweils für eine Menge von Attributwerten eines Tupels stehen. Das Zeichen "$\rightarrow$" bedeutet, dass die Attributwerte der linken Seite (Menge {X}), oft abgekürzt mit LHS (engl.: Left Hand Side), eindeutig die Attributwerte der rechten Seite (Menge {Y}), abgekürzt mit RHS (engl: Right Hand Side) bestimmen. {Y} ist somit funktional von {X} abhängig. Vereinfacht wird oft die abgekürzte Schreibweise $X \rightarrow Y$ verwendet. [3]

Eine erweiterte Form von Funktionalen Abhängigkeiten ist eine mehrwertige Abhängigkeit (engl.: Multivalued Dependency, abgekürzt MVD). Sie wird definiert als $\sigma = X \twoheadrightarrow Y$, wobei "$\twoheadrightarrow$" für eine mehrwertige Abhängigkeit steht. Sie besagt, dass einem Attributwert von X eine Menge von Y-Werten zugeordnet wird und Y somit mehrwertig abhängig von X ist. [31]

3.2.2 Schlüssel

Eine weit verbreitete Form von Funktionalen Abhängigkeiten sind Schlüssel. Sie werden verwendet, um Datenobjekte (also Tupel in relationalen Datenbanken) eindeutig zu identifizieren. Ein Schlüssel kann definiert werden als $\sigma = \{X\} \rightarrow \{U\}$, wobei {U} eine Menge von Attributen ist und {X} ist eine Teilmenge von {U}. [3]

[7] Die einzelnen Formen werden im Folgenden abgekürzt mit FDs, Keys, InclDs und FKs.

3.2.3 Inklusionsabhängigkeiten

Eine weitere wichtige Form von Integritätsbedingungen sind Inklusionsabhängigkeiten. Sie bezeichnen eine Teilmengenbeziehung zwischen zwei Folgen von Attributen und werden definiert als $\sigma = R[A_1, \cdots, A_m] \subseteq S[B_1, \cdots, B_m]$. *R* und *S* bezeichnen hierbei zwei (möglicherweise gleiche) Relationen und $[A_1, \cdots, A_m]$ bzw. $[B_1, \cdots, B_m]$ sind unterschiedliche Attribute in *R* bzw. *S*. [3]

3.2.4 Fremdschlüssel

Eine Form von Inklusionsabhängigkeiten sind Fremdschlüssel. Hierbei müssen die Attribute $[B_1, \cdots, B_m]$ als Primärschlüssel[8] der Relation *S* definiert sein. Fremdschlüssel sind somit eine Kombination von Inklusionsabhängigkeiten und Schlüsselbedingungen. Sie können verwendet werden, um auf Daten anderer (oder derselben) Relationen zu verweisen. [70]

3.2.5 Weitere Formen

Neben den soeben vorgestellten Formen von Integritätsbedingungen wurden noch weitere entwickelt. Beispiele dafür sind Join Dependencies, Embedded Dependencies, Mutual Dependencies oder Template Dependencies. Ein Überblick zu den weiteren Formen von Integritätsbedingungen kann u.a. in [3, 30, 31, 62] gefunden werden.

3.3 Integritätsbedingungen in XML

Ebenso wie bei relationalen Datenbanksystemen sind Integritätsbedingungen auch für XML von großer Bedeutung. Neben den bereits erwähnten Funktionen von Integritätsbedingungen in relationalen Datenbanken, erfüllen Integritätsbedingungen in XML noch weitere Aufgaben. Diese betreffen beispielsweise den Erhalt der originalen Datensemantik beim Datenaustausch oder das Vermeiden von Inkonsistenzen bei der Datenintegration. [33]

Aufgrund der flexiblen und gleichzeitig auch sehr komplexen und hierarchischen Datenstruktur von XML ist die Definition von XML Constraints schwieriger als beispielsweise relationale Constraints. Dazu kommt, dass die Entwicklung eines einheitlichen Systems

[8] Ein Primärschlüssel wird in [46] definiert als "ein Schlüssel, der in die Dateiorganisation direkt einbezogen wird und daher für direkte Zugriffe effizient benutzt werden kann."

für XML Integritätsbedingungen vom W3C vernachlässigt wurde. Das führte dazu, dass viele verschiedene theoretische Ansätze für XML Integritätsbedingungen entwickelt wurden. [35] Diese verschiedenen Ansätze, deren Unterschiede, Möglichkeiten sowie Vor- und Nachteile werden in den folgenden Kapiteln beschrieben. Dazu wird zuerst auf die Einteilung der Ansätze in verschiedene Kategorien gemäß Karlinger [49] eingegangen und anschließend die im weiteren Verlauf dieser Arbeit verwendete Syntax vorgestellt und definiert.

3.3.1 Kategorien von XML Integritätsbedingungen

XML Integritätsbedingungen können lt. Karlinger [49] in vier Kategorien unterteilt werden: Schema-Constraints, Pfad-Constraints, komplexe Constraints sowie wertbasierte Constraints. Im Folgenden wird ein Überblick zu diesem Thema gegeben. Genauere Informationen können in [49] nachgelesen werden.

Die erste Kategorie von XML Integritätsbedingungen sind sogenannte Schema-Constraints. Sie werden mittels Schema Spezifikationen definiert. Beispiele dafür sind die Einschränkung der Knotennamen auf eine bestimme Menge von Labels, das Vorkommen und die Kardinalität von Knoten, `min`- und `max`-Werte usw. [49]

Als zweite Kategorie werden Pfad-Constraints bezeichnet. Hierbei werden Datenobjekte über den Pfad über den sie erreichbar sind definiert. Die einzelnen Knoten werden somit anhand ihrer Pfade verglichen. XML Dokumente werden dazu meist als Graphen anstatt als Bäume dargestellt. [49]

Die nächste Kategorie von Integritätsbedingungen sind komplexe Constraints. Sie werden mit Hilfe von komplexen Regeln definiert und arbeiten ähnlich wie SQL-Abfragen in relationalen Datenbanken. Der Zugriff auf die Daten erfolgt meist mit Hilfe von mächtigen Abfragesprachen wie beispielsweise XPath [12] oder XQuery [16]. [49]

Die letzte Kategorie sind wertbasierte Constraints, die die jeweiligen Datenwerte zum Spezifizieren der Integritätsbedingungen verwendet. Dabei werden die Knoten im XML Dokument zuerst identifiziert (z.B. mittels einer Pfadsprache), um anschließend deren Werte zu vergleichen. [49]

3.3.2 Syntax

Die Syntax von Integritätsbedingungen in XML ist ähnlich aufgebaut, wie die Syntax von Constraints in relationalen Datenbanken. Im weiteren Verlauf dieser Diplomarbeit wird folgende Syntax verwendet:

Eine Funktionale Abhängigkeit wird definiert als $\sigma = (S, \{F_1, \cdots, F_n\} \rightarrow \{F'_1, \cdots, F'_n\})$ [24]. S und F stehen hierbei für die von XML Schema [32] übernommenen Bezeichnungen *Selector* und *Field*. Mit dem *Selector*-Pfad wird eine Menge von Elementen ausgewählt, deren Eigenschaften als *Fields* bezeichnet werden, wobei die mit den *Field*-Pfaden $\{F_1, \cdots, F_n\}$ zu erreichenden Knotenmengen die mit den *Field*-Pfaden $\{F'_1, \cdots, F'_n\}$ zu erreichenden Knotenmenge bestimmt. [24]

Schlüssel werden in XML definiert als $\sigma = (S, \{F_1, \cdots, F_n\})$. Dabei wird mit S die zu identifizierende Menge an Knoten ausgewählt und die Pfade $\{F_1, \cdots, F_n\}$ führen zu den identifizierenden Knoten. [49]

Um Inklusionsabhängigkeiten in XML zu definieren wird die Syntax $\sigma = (S, [F_1, \cdots, F_n]) \subseteq (S', [F'_1, \cdots, F'_n])$ verwendet. Hierbei steht $[F_1, \cdots, F_n]$ in einer Teilmengenbeziehung zu $[F'_1, \cdots, F'_n]$. Anstatt Mengen (welche ungeordnet sind) werden bei Inklusionsabhängigkeiten (geordnete) Listen von *Fields* verwendet, um zu gewährleisten, dass die einzelnen Knoten der LHS mit den zugehörigen Knoten der RHS verglichen werden. [49]

Bei Fremdschlüssel wird wiederum dieselbe Syntax verwendet wie bei Inklusionsabhängigkeiten.

3.4 Rahmen dieser Diplomarbeit

Diese Diplomarbeit beschränkt sich auf den Vergleich von wertbasierten Constraints. Ein Grund dafür ist, dass die einzelnen Kategorien von Integritätsbedingungen sehr unterschiedlich aufgebaut und ihre Anwendungsbereiche verschieden sind. Um ein einheitliches System zum Vergleich von IC-Ansätzen aller vier Kategorien zu finden, müssten Gemeinsamkeiten gefunden werden, die praktisch nicht vorhanden sind.

Des Weiteren wird eine Einschränkung auf wertbasierte Ansätze als sinnvoll erachtet, da durch den Vergleich von Datenobjekten aufgrund ihrer Datenwerte die Definition der weitverbreitetsten Constraint-Formen ermöglicht wird. Diese sind – ebenso wie für rela-

tionalen Datenbanken – Funktionale Abhängigkeiten, Schlüsselbedingungen Inklusionsabhängigkeiten und Fremdschlüssel. [9]

In den Kapiteln 5 bis 14 werden verschiedene Ansätze für XML Integritätsbedingungen vorgestellt, die sich teilweise erheblich voneinander unterscheiden. Um diese Unterschiede feststellen und beschreiben zu können, wird in Kapitel 4 eine Liste von Vergleichsmerkmalen vorgestellt, welche die im Allgemeinen als wichtig erachteten Eigenschaften bzw. Methoden von XML Integritätsbedingungen beschreibt. Aufbauend darauf werden einzelne Ansätze anhand dieser Merkmale analysiert und verglichen.

Kapitel 4

Vergleichsmerkmale

In diesem Kapitel wird eine Liste von Vergleichsmerkmalen zur Analyse von Ansätzen wertbasierter XML Integritätsbedingungen vorgestellt. Diese Merkmale bieten eine Möglichkeit, die große Anzahl vorhandener IC-Ansätzen zu analysieren und gegenüberzustellen. Sie betreffen die Bereiche Schemaspezifikation, Selektion der Knoten, Baummodell, Arität, Wertvergleich, Geltungsbereich von Constraints und Eigenschaften semistrukturierter Daten. Im Abschnitt 4.2 werden anschließend die einzelnen Merkmale zusammengefasst und die sich daraus ergebende Liste der Vergleichsmerkmale vorgestellt. Abschließend wird im Abschnitt 4.3 auf Analysen anderer Autoren zu diesem Thema eingegangen.

4.1 Beschreibung der Vergleichsmerkmale

Um die verschiedenen IC-Ansätze vergleichen zu können, muss zuerst ein Rahmen geschaffen werden, anhand dessen der Vergleich stattfindet. Es wurden bereits sehr viele verschiedene Ansätze für XML Constraints entwickelt, welche sich aufgrund verschiedener Eigenschaften, Feinheiten, Ausrichtungen, Methoden und Anwendungsbereichen unterscheiden [33, 35].

Die Schwierigkeit, die einzelnen Ansätze vergleichen zu können, liegt darin, deren Gemeinsamkeiten herauszufinden und somit eine Vergleichsbasis zu finden. Dieser Aufgabe widmeten auch andere Autoren ihre Aufmerksamkeit. Der Fokus lag dabei jedoch meist auf dem Vergleich des eigenen Ansatzes mit anderen Ansätzen, sodass dieser oft auf jene Bereiche beschränkt war, die im Ansatz selbst auch bearbeitet wurden (siehe Abschnitt 4.3).

In den folgenden Abschnitten werden die einzelnen Vergleichsmerkmale vorgestellt und deren Wichtigkeit erläutert. Es wird darauf hingewiesen, dass sich diese Analyse auf die wichtigsten Formen wertbasierter Constraints konzentriert und somit auf den Vergleich von Ansätzen für Funktionale Abhängigkeiten, Schlüssel und Fremdschlüssel sowie Inklusionsabhängigkeiten beschränkt (siehe Abschnitt 3).

Aufbauend darauf werden, mit Ausnahme des ersten Vergleichsmerkmals, Kontrollbeispiele angeführt, um herauszufinden, ob die einzelnen IC-Ansätze diese Merkmale erfüllen. Dafür ist es teilweise notwendig, dass für die verschiedenen Formen von Integritätsbedingungen (FDs, Keys, FKs und InclDs) jeweils eigene Beispiele angegeben werden. Für die Kontrollbeispiele werden verschieden XML Bäume angeführt, für die die Constraints im Sinne einer semantisch korrekten Umsetzung erfüllt bzw. nicht erfüllt sein sollen. Das gewünschte Ergebnis wird anschließend für jedes Kontrollbeispiel angegeben. Zu beachten ist außerdem, dass für Fremdschlüssel dieselben Kontrollbeispiele wie für Inklusionsabhängigkeiten verwendet werden. Der Grund dafür ist, dass Fremdschlüssel eine Form von Inklusionsabhängigkeiten sind, die jedoch voraussetzen, dass die identifizierenden Datenobjekte bereits als Primärschlüssel für andere Objekte definiert wurden [3].

Die Liste besteht aus 11 Vergleichsmerkmalen. Als erstes wird im Abschnitt 4.1.1 darauf eingegangen, ob ein IC-Ansatz an eine bestimmte Schemasprache gebunden ist. Anschließend wird im Abschnitt 4.1.2 die Selektionsmöglichkeit der Knoten analysiert, also der Zugriff auf die Daten mittels einer Pfadsprache.

Im Abschnitt 4.1.3 wird auf das vom Ansatz verwendete Baummodell eingegangen, insbesondere auf das Einhalten der Dokumentordnung und das Vorhandensein von Gemischtem Inhalt in XML Dokumenten.

Der Abschnitt 4.1.4 beschäftigt sich mit der Arität eines Constraints, d.h. ob es vom IC-Ansatz ermöglicht wird, n-äre Constraints zu definieren.

Als nächstes wird im Abschnitt 4.1.5 auf die verwendete Form des Wertvergleichs eingegangen. Diese betreffen den einfachen Wertvergleich, den Wertvergleich basierend auf der Verkettung von Textwerten (wenn in einem XML Dokument Elemente mit gemischtem Inhalt vorhanden sind) und dem baumbasierten Wertvergleich.

Abschnitt 4.1.6 geht auf den Geltungsbereich von Integritätsbedingungen ein und die Möglichkeit, ob Constraints auf bestimmte Bereiche eines XML Dokuments beschränkt werden können.

Abschließend werden im Abschnitt 4.1.7 zwei wichtige Eigenschaften semistrukturierter Daten im Zusammenhang mit Constraints analysiert: die Möglichkeiten, dass Datenobjekte in XML Dokumenten nicht oder aber mehrfach vorhanden sind.

4.1.1 Schema Spezifikation

Die Grundidee bei der Entwicklung von XML war es, ein flexibles Datenformat zu schaffen, um dadurch einen einfachen Datenaustausch zwischen unterschiedlichen Plattformen und Programmiersprachen zu ermöglichen. Um diese Flexibilität zu gewährleisten, wurden die syntaktischen Anforderungen an ein XML Dokument so gering wie möglich gehalten. Sie werden anhand der Regeln der Wohlgeformtheit definiert. Die einzige Voraussetzung an ein XML Dokument ist somit, dass es wohlgeformt sein muss (siehe Abschnitt 2.2.1). [66]

Darüber hinaus kann ein Schema für ein XML Dokument definiert werden (siehe Abschnitt 2.4). Darin können, neben allgemeinen Constraints die v. a. den strukturellen Aufbau des Dokuments betreffen, auch semantische Constraints wie beispielsweise Schlüssel und Fremdschlüssel definiert werden. Ein Beispiel dafür ist die Verwendung der `ID`- und `IDREF`-Attribute in DTDs [17]. [66]

Obwohl ein Schema große Vorteile bei der Entwicklung und Strukturierung von XML Dokumenten bringen kann, können auch Nachteile damit verbunden sein. Diese betreffen vor allem die Flexibilität von Daten und den Datenaustausch. Werden beispielsweise Daten aus unterschiedlichen Quellen mit verschiedenen Schemata ausgetauscht, kann dies den Datenaustausch erschweren oder im Extremfall sogar unmöglich machen. [33]

Dennoch ist die Verwendung von Schemata weit verbreitet und wird wahrscheinlich gerade deshalb oft als Basis für die Entwicklung eines IC-Ansatzes verwendet. Dabei gibt es Ansätze, die nur unter der Voraussetzung angewendet werden können, dass eine Schemaspezifikation vorhanden ist. [43]

Des Weiteren gibt es Ansätze, die direkt auf bestimmte Konstrukte einer Schemasprache aufgebaut sind und somit nur gemeinsam mit dieser Sprache bzw. einer Schemasprache, die diese Funktionen unterstützt, angewendet werden können. Ein Beispiel dafür ist der Ansatz von Chen et al. [23], deren Definition einer funktionalen Abhängigkeit auf die Verwendung von komplexen Elementtypen in XML Schema [15, 32, 76] basiert. Eine Anwendung dieses Ansatzes zusammen mit einer einfacheren Schemasprache wie beispielsweise DTD [17] wäre somit nicht möglich, da mit DTDs keine komplexen Elementtypen

definiert werden können. [23] Ebenso wäre es nicht möglich, den Ansatz bei einem XML Dokument anzuwenden, dass gar kein Schema verwendet [79].

Darüber hinaus ist es bei Typ-basierten Schemasprachen wie DTD [17] oder XML Schema [15, 32, 76] auch möglich, dass sie die Verwendung von Integritätsbedingungen negativ beeinflussen [22]. Das folgende Beispiel macht dies deutlich (entnommen aus [18]):

Beispiel 4.1: *Für ein XML Dokument wird folgende DTD definiert:*
`<!ELEMENT foo (X, X)>`
`<!ELEMENT X (empty)>`
Jedes Element `foo` *muss somit zwei verschiedene Elemente* `X` *beinhalten. Darüber hinaus wird folgender Schlüssel definiert* σ `= (X,` $\emptyset$`)`*. Es handelt sich dabei um einen strukturellen Schlüssel (engl.: Structural Key), d.h. ein Schlüssel mit einer leeren Menge an identifizierenden Attributen. Strukturelle Schlüssel werden verwendet, um zu definieren, dass ein Datenelement zwar vorhanden sein muss, jedoch unabhängig von Datenwerten identifiziert wird [42]. Der Schlüssel* σ `= (X,` $\emptyset$`)` *besagt, dass sich, wenn vorhanden, genau ein eindeutig zu identifizierendes Element* `X` *in* `foo` *befinden muss, wohingegen die DTD fordert, dass jedes Element* `foo` *zwei verschiedene Elemente* `X` *beinhaltet. Es gibt somit kein XML Dokument das sowohl der DTD entspricht als auch den Schlüssel erfüllt.*

Das Beispiel zeigt, dass die zwingende Verwendung einer Schemasprachen bei einem Ansatz die Definition der Integritätsbedingungen erschweren kann und teilweise sogar unmöglich macht.

Die soeben genannten Probleme, aber auch die Tatsache, dass eine Grundidee von semistrukturierten Daten die Unabhängigkeit von einem Schema ist (siehe Abschnitt 2.2.2), führen zur Definition des ersten Vergleichsmerkmals:

Vergleichsmerkmal 1: *Der Ansatz ist nicht an eine Schemasprache gebunden.*

Dieses Merkmale kann ohne Kontrollbeispiel überprüft werden, weshalb auf die Angabe eigener Beispiele verzichtet wird.

4.1.2 Ausdruckskraft der Pfadsprache

Um Datenobjekte in einem XML Dokument abrufen zu können, können Pfadsprachen verwendet werden. Die bekannteste Pfadsprache XPath [12] und eine kleine Auswahl an

möglichen Pfadfunktionen wurden im Abschnitt 2.3 vorgestellt. Zweifelsohne erweitern solche Funktionen die Ausdruckskraft der Pfadsprachen und ermöglichen somit die Spezifikation von komplexeren Anwendungen. [41]

Pfadsprachen können auch zur Definition von Integritätsbedingungen verwendet werden, um auf die jeweiligen Knoten und deren Werte zugreifen zu können. Bei manchen IC-Ansätzen werden teilweise bestimmte Funktionen einer Pfadsprache vorausgesetzt, ohne die eine Definition der Constraints gemäß diesem Ansatz nicht oder nur teilweise möglich ist. Diese Anforderungen reichen von einfachen Abwärtspfaden bis hin zu komplexen und teilweise sehr komplizierten Abfrageoptionen, welche eine effiziente Anwendung teilweise erschweren und somit eine praktische Umsetzbarkeit erheblich beeinträchtigen können. [41] Andere Ansätze gehen hingegen nur von Grundfunktionen einer Pfadsprache aus (z.B. die Ansätze in [6, 7] bzw. [86]).

Da die Bandbreite solcher Funktionen sehr groß ist, soll hierauf nicht im Detail eingegangen werden. Vielmehr werden im folgenden Abschnitt jene Eigenschaften von Pfadsprachen herausgearbeitet, die für eine große Anzahl von Anwendungsbereichen von Integritätsbedingungen notwendig sind.

Abwärtspfade

Die wichtigste Funktion einer Pfadsprache ist das Definieren einfacher Abwärtspfade. Dies wird mit Hilfe von Verkettungsoperatoren ermöglicht und wird als essentielle Funktion von Pfadsprachen und auch zur Definition von Constraints vorausgesetzt. [42]

Kleene Operator

Bei der Verwendung von Integritätsbedingungen wird zur Definition der Pfade häufig ein Kleene Operator verwendet. [41] Dafür gibt es mehrere Gründe.

Ein wichtiges Argument für die Verwendung eines Kleene Operators ist, dass dadurch die angegebenen Pfade erheblich verkürzt werden können und somit die Definition der Constraints vereinfacht wird. [41] Beispielsweise kann dadurch der Pfad `p = uni.lvas.lva.vt.@name` mit dem Kleene Operator verkürzt definiert werden als: `p = _*.vt.@name`.

Des Weiteren gibt es Constraints, die ohne Kleene Operatoren nicht definiert werden können [88], wie das folgende Beispiel zeigt:

Beispiel 4.2: *Angenommen ein XML Dokument repräsentiert den Inhalt eines Buches. Der*

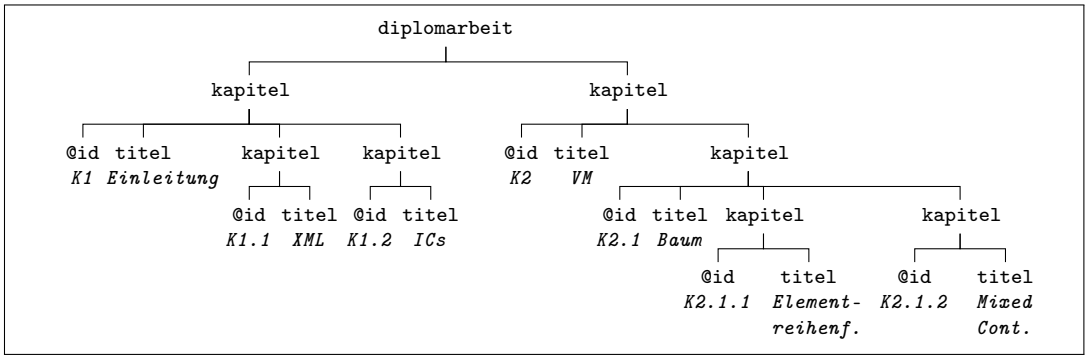

Abbildung 4.1: XML Baum zur Darstellung der Wichtigkeit von Kleene Operatoren

XML Baum wird in Abbildung 4.1 dargestellt. Das Element `buch` *besteht aus beliebig vielen Kapiteln, Unterkapiteln, Unterunterkapiteln usw. Die Anzahl der Hierarchiestufen ist dabei nicht begrenzt. Jedes Element* `kapitel` *hat ein Attribut* `id`*, das zur Identifikation des Kapitels verwendet wird. Der zugehörige Schlüssel lautet somit* $\sigma = ($`_*.kapitel, {@id}`$)$*. Da die Anzahl der Hierarchiestufen nicht beschränkt ist, könnte dieser Schlüssel ohne Kleene Operatoren nicht definiert werden.*

Pfade zu Elementknoten

Bei der Definition von Integritätsbedingungen soll es auch möglich sein, Pfade zu verwenden, die mit einem Elementknoten enden (anstatt mit einem Blattknoten, d.h. einem Attribut- oder Textknoten). Im Falle von Schlüsselbedingungen soll dies nicht nur auf die zu identifizierenden Knoten beschränkt sein, sondern auch für die identifizierenden Knoten erlaubt werden. Gleiches gilt bei der Definition von Fremdschlüsseln, Funktionalen Abhängigkeiten und Inklusionsabhängigkeiten. [18]

Dies ist v.a. für Constraints relevant, deren Field-Knoten mittels Wertvergleich der Elemente unterschieden werden. Dies wird im Abschnitt 4.1.5 nochmals verdeutlicht.

Es gibt dabei Ansätze, in denen der Wert eines Elements dessen Textwert ist. Wird beispielsweise der Pfad `uni.lvas.lva.titel` definiert, wird als Wert des Elements `titel` dessen Textwert zurückgegeben (z.B. *VO DKE*). In anderen Ansätzen ist der Wert eines Elements dessen Identität und der Textwert wird über den darunter liegenden Textknoten aufgerufen. Der Pfad lautet dann: `uni.lvas.lva.titel.S`.

Im weiteren Verlauf dieser Diplomarbeit bezeichnet das Elementlabel somit den Elementknoten, während das Label `S` zum Zugriff auf den Textwert verwendet wird.

Andere Formen des Wertvergleichs von Elementknoten sind der Wertvergleich von gemischtem Inhalt und der baumbasierte Wertvergleich. Hierauf wird im Abschnitt 4.1.5 eingegangen.

Zusammenfassung

Zusammenfassend kann gesagt werden, dass für die Definition von Constraints eine gewisse Ausdruckskraft der Pfadsprache vorhanden sein soll. Dabei ist es wichtig abzuwägen, welche Funktionen wichtig bzw. notwendig sind, und auf welche (z.B. wegen ihrer Komplexität) gegebenenfalls verzichtet werden kann. Aufgrund der oben angegebenen Punkte werden folgende Funktionen als wichtig erachtet:

- Einfache Abwärtspfade mittels Verkettungsoperatoren
- Kleene Operatoren
- Pfad kann in Element-, Attribut- oder Textknoten enden

Da die Definition von Abwärtspfaden eine grundlegende Funktion von Pfadsprachen darstellt und somit vorausgesetzt wird, wird diese bei der Formulierung und Kontrolle des nächsten Vergleichsmerkmals nicht weiter berücksichtigt. Dieses lautet somit:

Vergleichsmerkmal 2: *Der Ansatz unterstützt die Verwendung eines Kleene Operators sowie Pfade, die mit Elementknoten enden.*

Um zu kontrollieren, ob dieses Vergleichsmerkmal von einem IC-Ansatz erfüllt wird, werden nachfolgende Kontrollbeispiele anhand der XML Bäume in den Abbildungen 4.2 und 4.3 getestet. Der Wertvergleich ist dabei auf einfache Textwerte beschränkt. Andere Möglichkeiten des Wertvergleichs siehe Abschnitt 4.1.5.

Kontrollbeispiel 2.1 - XML Schlüssel:
Ein Vortragender (Element `vt`*) wird mit seinem Namen (Element* `name`*) identifiziert. Der Schlüssel lautet somit:* σ `= (_*.vt, {name})`*. Durch die Definition mittels Kleene Operator gilt dieser Schlüssel für alle Elemente* `vt`*, egal wo im Dokument sie sich befinden. Ohne Kleene Operator könnte dies nicht definiert werden oder lediglich mit der Definition von drei Schlüsseln* σ `= (uni.inst.lva.vt, {name})`*,* σ `= (uni.lvas.lva.vt, {name})` *und* σ `= (uni.vortragende. vt, {name})` *teilweise abgebildet werden. Dadurch wird jedoch definiert, dass nur innerhalb des Teilbaums* `inst`*,* `lvas` *bzw.* `vortragende` *ein Vortragender eindeutig identifiziert werden kann. Dies weicht von der*

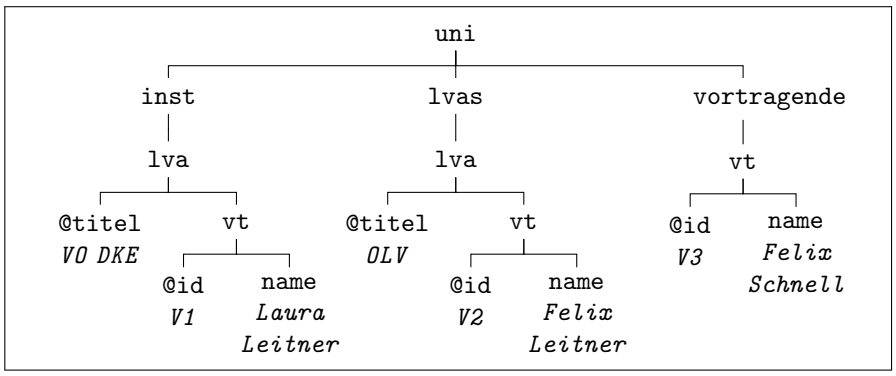

Abbildung 4.2: XML Baum 1 zur Überprüfung der Pfadfunktionen

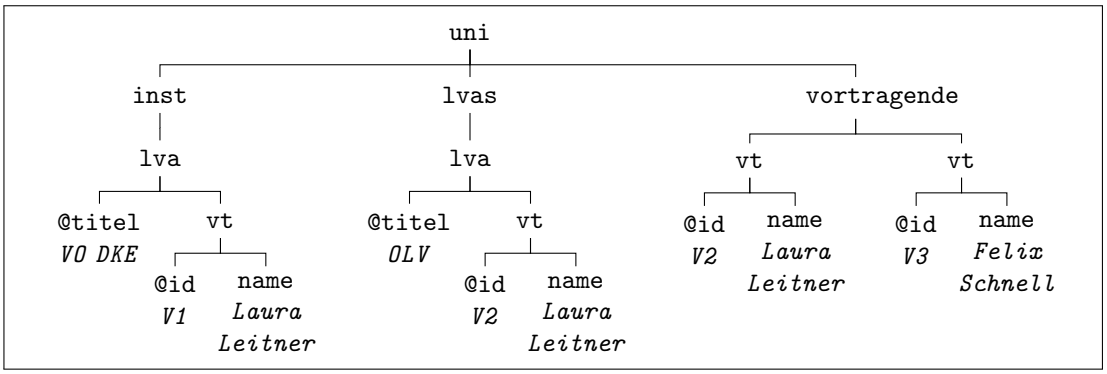

Abbildung 4.3: XML Baum 2 zur Überprüfung der Pfadfunktionen

zu definierenden Bedingung ab. Wird nun die mittels Kleene Operator definierte Schlüsselbedingung σ = (_.vt, {name}) verwendet, ist diese für den XML Baum in Abbildung 4.2*[9] *erfüllt, da die drei Vortragenden unterschiedliche Namen haben und somit eindeutig identifiziert werden können. Dies wird durch folgende Aufstellung ersichtlich:*

Teilbaum	*vt*	*name*
inst	*1*	*Laura Leitner*
lvas	*2*	*Felix Leitner*
vortragende	*3*	*Felix Schnell*

Kontrollbeispiel 2.2 - XML Schlüssel:
Der im Kontrollbeispiel 2.1. angegebenen Schlüssel σ = (_.vt, {name}) wird nun für den XML Baum in Abbildung 4.3 verwendet. Für diesen XML Baum ist die Schlüsselbedingung nicht erfüllt, da drei Vortragende mit demselben Namen im Baum vorhanden sind und somit nicht eindeutig identifiziert werden können. Folgende Auflistung verdeutlicht dies:*

Teilbaum	*vt*	*name*
inst	*1*	*Laura Leitner*
lvas	*2*	*Laura Leitner*
vortragende	*3*	*Laura Leitner*
vortragende	*4*	*Felix Schnell*

Würden anstatt dem Schlüssel σ = (_.vt, {name}) die drei ohne Kleene Operator definierten Schlüsseln σ = (uni.inst.lva.vt, {name}), σ = (uni.lvas.lva.vt, {name}) und σ = (uni.vortragende. vt, {name}) verwendet werden, würde dies zu einem falschen Ergebnis führen. Alle drei Schlüsselbedingungen wären in diesem Fall erfüllt, da die name-Werte der Teilbäume inst, lvas und vortragende unabhängig voneinander geprüft und die Vortragenden somit eindeutig identifiziert werden könnten. Dies entspricht jedoch nicht dem gewünschten Ergebnis, sodass eine korrekte Umsetzung ohne Kleene Operator nicht möglich ist.*

Kontrollbeispiel 2.3 - XML FDs:
Die ID eines Vortragenden (Attribut @id) wird bestimmt durch seinen Namen. Die Funktionale Abhängigkeit lautet somit σ = (_.vt, {name} → {@id}). Angewendet auf*

[9] *OLV* steht hierbei für eine Orientierungslehrveranstaltung, welche unabhängig von einem Institut abgehalten wird

den XML Baum in Abbildung 4.2 ist diese Bedingung erfüllt, da die drei Vortragenden mit unterschiedlichem Namen auch verschiedene `id`-Werte haben:

Teilbaum	`name`	`@id`
inst	*Laura Leitner*	*V1*
lvas	*Felix Leitner*	*V2*
vortragende	*Felix Schnell*	*V3*

Ohne Kleene Operator könnte diese Funktionale Abhängigkeit nicht korrekt abgebildet werden. Es könnten lediglich die drei Funktionalen Abhängigkeiten σ = (`inst.vt`, `{name}` $\rightarrow$ `{@id}`), σ = (`lvas.vt`, `{name}` $\rightarrow$ `{@id}`) und σ = (`vortragende.vt`, `{name}` $\rightarrow$ `{@id}`) definiert werden. Diese würde jedoch zu einer Prüfung in den drei Teilbäumen `inst`, `lvas` und `vortragende` unabhängig voneinander führen und eine einheitliche Prüfung im gesamten XML Baum wäre nicht möglich.

Kontrollbeispiel 2.4 - XML FDs:
Die Funktionale Abhängigkeit σ = (`_*.vt`, `{name}` $\rightarrow$ `{@id}`) wird nun für den XML Baum in Abbildung 4.3 verwendet. Hier ist die Bedingung nicht erfüllt, da die Vortragenden mit dem Namen `Laura Leitner` verschiedene `id`-Werte haben:

Teilbaum	`name`	`@id`
inst	*Laura Leitner*	*V1*
lvas	*Laura Leitner*	*V2*
vortragende	*Laura Leitner*	*V2*
vortragende	*Felix Schnell*	*V3*

Würden hingegen die drei Funktionalen Abhängigkeiten σ = (`inst.vt`, `{name}` $\rightarrow$ `{@id}`), σ = (`lvas.vt`, `{name}` $\rightarrow$ `{@id}`) und σ = (`vortragende.vt`, `{name}` $\rightarrow$ `{@id}`) verwendet werden, könnten die IDs eindeutig bestimmt werden, da die `name`-Werte der drei Teilbäume unabhängig voneinander geprüft werden. Dies entspricht jedoch nicht dem gewünschten Ergebnis.

Kontrollbeispiel 2.5 - XML InclDs:
Der Name (Element `name`) eines Vortragenden einer Lehrveranstaltung (Teilbäume `inst` und `lvas`) muss gleichzeitig im Teilbaum `vortragende` vorhanden sein. Die Inklusionsabhängigkeit lautet somit: σ = (`_*.lva.vt`, `[name]`) $\subseteq$ (`_*.vortragende.vt`, `[name]`). Angewendet auf den XML Baum in Abbildung 4.2 ist diese Bedingung nicht erfüllt, da der Name Laura Leitner im Teilbaum `inst` vorhanden ist, im Teilbaum `vortragende` jedoch nicht.

Teilbaum inst	*Teilbaum lvas*	*Teilbaum vortragende*
name	*name*	*name*
Laura Leitner	*Felix Schnell*	*Felix Schnell*

Diese Inklusionsabhängigkeit kann nur mit Kleene Operator korrekt definiert werden. Würden stattdessen zwei Inklusionsabhängigkeiten ohne Kleene Operator definiert werden, würden diese lauten σ *= (uni.inst.lva.vt, [name])* $\subseteq$ *(uni.vortragende.vt, [name]) und* σ *= (uni.lvas.lva.vt, [name])* $\subseteq$ *(uni.vortragende.vt, [name]). Hierbei wäre zumindest die zweite Inklusionsabhängigkeit erfüllt, da der Wert Felix Schnell sowohl im Teilbaum lvas als auch im Teilbaum vortragende vorhanden ist. Dies würde jedoch zu einem falschen Ergebnis führen.*

Kontrollbeispiel 2.6 - XML InclDs:
Die Inklusionsabhängigkeit lautet wiederum σ *= (_*.lva.vt, [name])* $\subseteq$ *(_*.vortragende.vt, [name]) und wird nun auf den XML Baum in Abbildung 4.3 angewendet. Hier ist die Bedingung erfüllt, weil die Namen der Vortragenden in den Teilbäumen inst und lvas auch im Teilbaum vortragende vorhanden sind, wie folgende Auflistung zeigt:*

Teilbaum inst	*Teilbaum lvas*	*Teilbaum vortragende*
name	*name*	*name*
Laura Leitner	*Laura Leitner*	*Laura Leitner* *Felix Schnell*

Auch hier wäre eine Definition von zwei Inklusionsabhängigkeiten ohne Kleene Operator unzulässig, weil dadurch die Teilbäume inst und lvas einzeln geprüft werden würden.

4.1.3 Baummodell

Wie bereits in Kapitel 2.2.1 beschrieben wurde, muss ein XML Dokument bestimmten Regeln entsprechen, um wohlgeformt zu sein. Darüber hinaus kann ein bestimmtes Baummodell, wie beispielsweise das XML Information Set [26] oder das Document Object Model (DOM) [56], verwendet werden.

Die Ansätze für XML Integritätsbedingungen beruhen in der Regel ebenfalls auf einem Baummodell. Zwei wichtige Bereiche sind dabei die Dokumentordnung (d.h. das Einhalten der Reihenfolge, in der die Knoten aufgelistet sind) sowie das Vorhandensein von gemischtem Inhalt [53]. Auf diese wird nun näher eingegangen.

Dokumentordnung

Wie bereits mehrfach erwähnt wurde, kann ein Element Unterelemente beinhalten, welche gemäß der XML Spezifikation [17] korrekt verschachtelt sein müssen. Elemente mit denselben Eltern-Knoten werden als Geschwister-Knoten bezeichnet. Dabei ist es erlaubt, dass mehrere Geschwister-Knoten dasselbe Label haben. Neben diesen Anforderungen des W3C wurden keine weiteren Regeln betreffend des Auftretens von Elementen für wohlgeformte XML Dokumente definiert. [17]

Im Gegensatz dazu, wird in vielen XML-Baummodellen explizit auf die Reihenfolge der Knoten eingegangen. Ein bekanntes Beispiel dafür ist das XML Information Set [26], dass die Kinder eines Elements in Form einer geordneten Liste definiert:

Definition 4.1: *The children of an element are: "An ordered list of child information items, in document order. This list contains element, processing instruction, unexpanded entity reference, character, and comment information items* [...]*." [26]*

Im Vergleich dazu werden die Attribute eines Elements als ungeordnete Menge angegeben:

Definition 4.2: *The attributes of an element are: "An unordered set of attribute information items, one for each of the attributes* [...] *of this element." [26]*

Weitere bekannte Datenmodelle, die die Dokumentordnung beachten, sind das DOM [56] oder das in XPath [12] und XQuery [16] verwendete Datenmodell [13].

Diese Eigenschaft weicht erheblich von den Regeln in traditionellen Datenbanksystemen ab. In relationalen Datenbanksystemen wird beispielsweise die Reihenfolge, in der die einzelnen Tupel gespeichert werden, nicht beachtet. [52] Es stellt sich nun die Frage, warum in XML Dokumenten die Reihenfolge als wichtig erachtet wird.

Ein wichtiger Aspekt in diesem Zusammenhang ist die Unterscheidung zwischen datenorientierten und dokumentorientierten XML Dokumenten. Da datenorientierte XML Dokumente ähnlich funktionieren wie beispielsweise relationale Datenbanken, ist die Reihenfolge in solchen XML Dokumenten oft nicht relevant. [52] Dies wird beispielsweise durch die XML Dokumente in Abbildung 4.4 deutlich. Es zeigt zwei datenorientierte XML Dokumente, die jeweils zwei `student`-Elemente beinhalten, welche jedoch in un-

(a)

```
<vortragende>

  <vortragender>
    <vn>Laura</vn>
    <nn>Leitner</nn>
   </vortragender>

   <vortragender>
     <vn>Felix</vn>
     <nn>Schnell</nn>
   </vortragender>

</vortragende>
```

(b)

```
<vortragende>

   <vortragender>
     <vn>Felix</vn>
     <nn>Schnell</nn>
   </vortragender>

   <vortragender>
     <vn>Laura</vn>
     <nn>Leitner</nn>
   </vortragender>

</vortragende>
```

Abbildung 4.4: Datenorientierte XML Dokumente mit unterschiedlicher Dokumentordnung

terschiedlicher Reihenfolge abgespeichert wurden. Es ist offensichtlich, dass die beiden XML Dokumente trotz unterschiedlicher Reihenfolge der Elemente inhaltlich äquivalent sind. Ein Vertauschen der Reihenfolge hätte somit keine negativen Auswirkungen.

Im Vergleich dazu kann die Reihenfolge der Elemente und Textwerte in dokumentorientierten XML Dokumenten von großer Bedeutung sein, um eine korrekte Wiedergabe der Wirklichkeit zu ermöglichen. [52] Dies soll durch die beiden XML Dokumente in Abbildung 4.5 verdeutlicht werden. Sie zeigen jeweils einen Ausschnitt eines Buches mit mehreren Kapiteln. Die einzelnen `kapitel`-Elemente werden jedoch in unterschiedlicher Reihenfolge gespeichert. Während das erste XML Dokument der Realität entspricht (in der Regel kommt die Einleitung vor der Zusammenfassung in einem Buch), führt das Vertauschen der Elementreihenfolge im zweiten XML Dokument zu einem falschen Ergebnis.

Teilweise ist es nicht möglich, ein XML Dokument eindeutig als datenorientiert oder dokumentorientiert zu bezeichnen. Vielmehr beinhalten XML Dokumente oft sowohl datenorientierten als auch dokumentorientierten Inhalt. [54] Aus diesem Grund soll die Einhaltung der Dokumentordnung bei sämtlichen XML Dokumenten gewährleistet sein.

Ein weiteres Argument, das für das Einhalten der Dokumentordnung spricht, ist, dass

(a)

```
<buch>

   <kapitel>
      Einleitung
  </kapitel>

  <kapitel>
    Zusammenfassung
  </kapitel>

</buch>
```

(b)

```
<buch>

   <kapitel>
     Zusammenfassung
  </kapitel>

  <kapitel>
    Einleitung
  </kapitel>

</buch>
```

Abbildung 4.5: Dokumentorientierte XML Dokumente mit unterschiedlicher Dokumentordnung

einige XML Werkzeuge diese Eigenschaft voraussetzen. Dies ist beispielsweise bei XPath [12] der Fall, wo ein Navigieren zwischen den (geordneten) Geschwister-Knoten möglich ist oder spezielle Funktionen, wie z.B. `last()`, verwendet werden können. Um diese nutzen zu können, müssen die einzelnen Knoten einer Ordnung unterliegen. [12]

Andere Werkzeuge, wie beispielsweise XML Schema [15, 32, 76], überlassen dem Entwickler die Entscheidung, ob die Dokumentordnung beachtet werden soll oder nicht. XML Schema stellt dafür zwei Funktionen zur Verfügung. Wird bei der Spezifikation eines Elements die Funktion `<all>` verwendet, können dessen Unterelemente in beliebiger Reihenfolge auftreten. Die Funktion `<sequence>` bedeutet hingegen, dass die Elemente genau in der angegebenen Reihenfolge vorkommen müssen. [32]

Im weiteren Verlauf dieser Diplomarbeit werden noch andere Anwendungsbereiche gezeigt, für die das Einhalten der Dokumentordnung notwendig ist. Von großer Bedeutung sind dabei die im Abschnitt 4.1.5 vorgestellten Formen des Wertvergleichs. Obwohl sie erst im weiteren Verlauf dieser Diplomarbeit beschrieben werden, sollen sie trotzdem als weiteres Argument dienen, das für die Einhaltung der Dokumentordnung spricht.

Das nächste Vergleichsmerkmal lautet somit:

Vergleichsmerkmal 3: *Der Ansatz verwendet ein Baummodell, das das Einhalten der Dokumentordnung gewährleistet.*

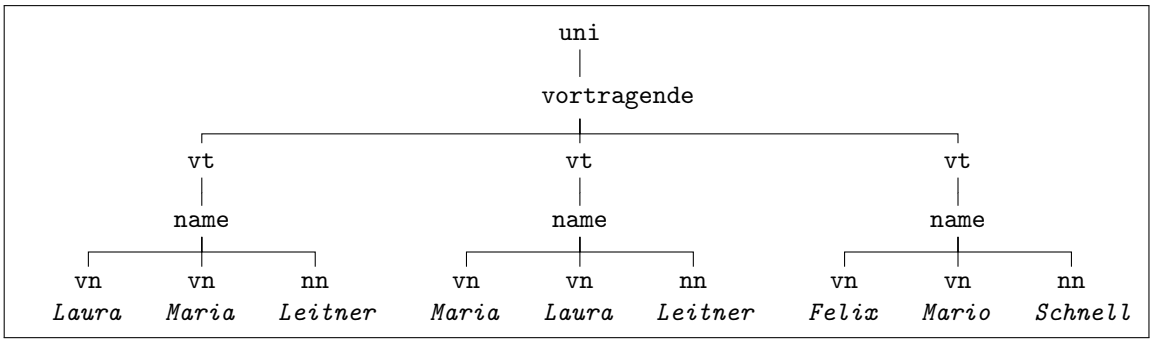

Abbildung 4.6: XML Baum zur Überprüfung der Dokumentordnung

Um überprüfen zu können, ob das Baummodell eines Ansatzes dieses Merkmal erfüllt, soll der im nachfolgenden Kontrollbeispiel beschriebene XML Baum abgebildet werden können.

Kontrollbeispiel 3.1 - Einhaltung der Dokumentordnung:
Der XML Baum in Abbildung 4.6 zeigt eine Liste von Vortragenden. Jedes Element `vt` *beinhaltet ein Unterelement* `name`*, das wiederum aus den Unterelementen* `vn` *und* `nn` *besteht. Die Namen der Vortragenden lauten somit "Laura Maria Leitner", "Maria Laura Leitner" und "Felix Mario Schnell". Bei den ersten beiden Vortragenden wird deutlich, dass die Anordnung der Elemente von Bedeutung ist. Es handelt sich dabei um zwei verschiedenen Vortragende, deren Namen nur dann korrekt wiedergegeben werden können, wenn die Elemente* `vn` *und* `nn` *korrekt angeordnet sind.*

Gemischter Inhalt

Als nächstes wird auf die Verwendung von gemischtem Inhalt eingegangen. Wie bereits mehrfach erwähnt wurde, bezeichnet der Inhalt eines Elements einerseits eine geordnete Liste von Unterelementen und andererseits etwaige Textwerte. Die Attribute eines Elements werden meist als dessen Eigenschaft gesehen und nicht als dessen Inhalt. Wird nun von gemischtem Inhalt gesprochen, bedeutet dies, dass ein Element sowohl Unterelemente als auch Textwerte beinhaltet. [53]

Die verschiedenen Baummodelle verwenden in diesem Zusammenhang unterschiedliche Vorgehensweisen. Während manche Modelle den gesamten Text eines Elements in einem einzigen Textknoten vereinen, bieten andere Modelle die Möglichkeit, den Text auf mehrere direkt und indirekt nachfolgende Textknoten zu verteilen. Der Inhalt eines Elements

ist dann die Verkettung aller nachfolgenden Textwerte. [53] Hierauf wird im Abschnitt 4.1.5 näher eingegangen.

Von großer Bedeutung ist gemischter Inhalt bei dokumentorientierten XML Dokumenten, da hierbei oft Textwerte (z.B. der Text eines Kapitels in einem Buch) gemeinsam mit deren Aufbau und Struktur (z.B. die Anordnung der Kapitel eines Buches) abgebildet werden. [53]

Des Weiteren ist für manche XML Werkzeuge die Verwendung von gemischten Inhalt unumgänglich. Dies ist beispielsweise bei XHTML [68] der Fall, eine auf XML basierte Erweiterung von HTML 4.01 [69]. Abbildung 4.7 zeigt einen Ausschnitt eines XHTML Dokuments, in dem eine Wegbeschreibung mittels gemischtem Inhalt in XHTML-Form abgebildet wird.

```
<html>
    <head>
        <title>Anreise</title>
    </head>
    <body>
        <h1>Wegbeschreibung</h1>
        <p>
            Bitte folgen Sie dem Straßenverlauf für
            <b>5 km </b>
            und biegen Sie danach rechts ab.
        </p>
    </body>
</html>
```

Abbildung 4.7: Wegbeschreibung in Form eines XHTML Dokuments

Die eben beschriebenen Anwendungsbereiche von gemischtem Inhalt sowie deren Wichtigkeit für bestimmte XML Werkzeuge führen zum nächsten Vergleichsmerkmal:

Vergleichsmerkmal 4: *Der Ansatz verwendet ein Baummodell, das gemischten Inhalt unterstützt.*

Um überprüfen zu können, ob dies von einem Ansatz ermöglicht wird, soll nachfolgendes Kontrollbeispiel korrekt umgesetzt werden können.

Kontrollbeispiel 4.1 - Unterstützung von gemischtem Inhalt:

Der XML Baum in Abbildung 4.8 zeigt zwei Elemente `vt` *deren Unterelemente* `name` *die Namen der Vortragenden enthalten. Der Name ergibt sich dabei aus der Verkettung aller direkt und indirekt nachfolgenden Textwerte. Somit lautet der Name der ersten Vortragenden "Dr. Laura Leitner" und der Name der zweiten Vortragenden "Dr. Laura Leitner sen."*[10]*. Um dies korrekt abbilden zu können, muss gemischter Inhalt erlaubt sein.*

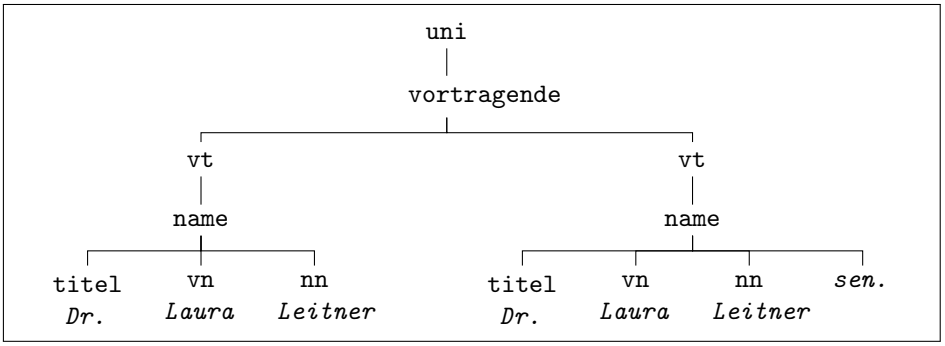

Abbildung 4.8: XML Baum zur Überprüfung von gemischtem Inhalt im Baummodell

4.1.4 Arität

Eine weitere wichtige Eigenschaft, die die Ausdruckskraft von Integritätsbedingungen erheblich beeinflusst, ist die Arität (Stelligkeit, engl.: arity). Bei Schlüsselbedingungen beschreibt die Arität die Anzahl der Werte (d.h. Element-, Attribut- oder Textwerte) die zum Identifizieren eines Knoten verwendet werden. Bei funktionalen Abhängigkeiten sind dies die Werte, die einen Knoten eindeutig bestimmen. Gleiches gilt für Inklusionsabhängigkeiten und Fremdschlüsseln. [3]

Wird genau ein Wert zur Definition des Constraints verwendet, wird er als unärer Constraint bezeichnet. Ein einfaches Beispiel wäre die Identifikation eines Elements mit genau einem ID-Attribut. Diese Form der Arität ist sehr einschränkend, da viele Constraints mit nur einem Wert nicht definiert werden können. Dies wäre beispielsweise der Fall, wenn eine Person mit ihrem Vor- und Nachnamen identifiziert werden soll, die Werte jedoch als zwei getrennte Attribut- oder Textwerte im XML Dokument abgespeichert sind. Um diesen Schlüssel definieren zu können, müssen zwei Werte (Vorname und Nachname) zur Identifikation verwendet werden. Der Schlüssel würde somit lauten: σ = `(person, {@vn, @nn})`. Diese Art von Constraints wird als n-ärer Constraint bezeichnet. [33]

[10] "sen." stellt hierbei einen Textknoten mit dem Wert "sen." dar

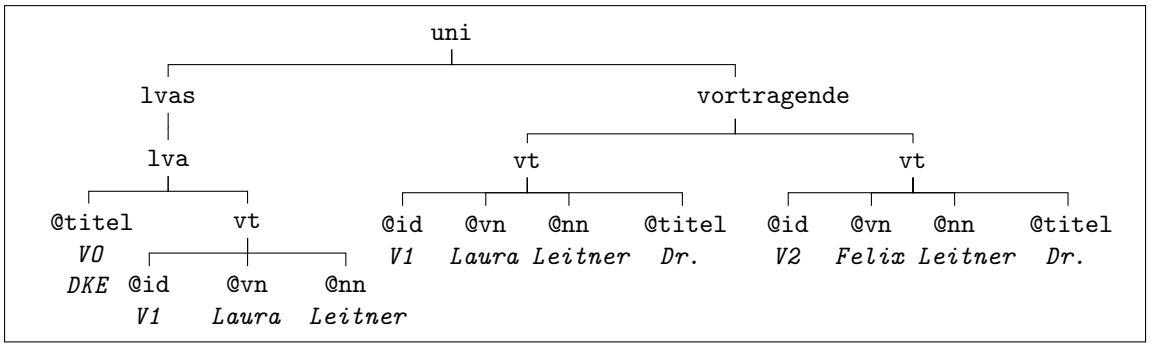

Abbildung 4.9: XML Baum 1 zur Überprüfung der Arität

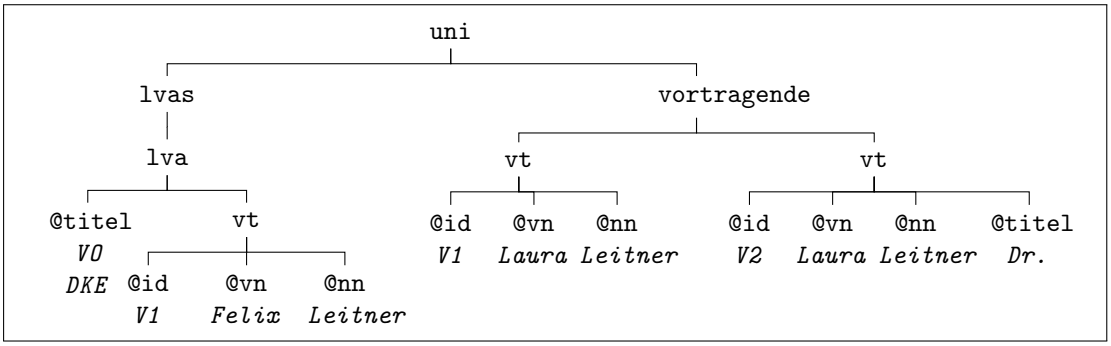

Abbildung 4.10: XML Baum 2 zur Überprüfung der Arität

N-äre Constraints haben eine viel höhere Ausdruckskraft als unäre und ermöglichen die Spezifikation von Bedingungen, die mit unären Constraints nicht definiert werden könnten. [33] Beispielsweise könnte der oben genannte Schlüssel als unärer Constraint nicht ausgedrückt werden, selbst wenn zwei unäre Constraints σ = (person, {@vn}) und σ = (person, {@nn})) verwendet werden würden.

Das nächste Vergleichsmerkmal lautet somit:

Vergleichsmerkmal 5: *Der Ansatz ermöglicht die Definition von n-ären Integritätsbedingungen.*

Um herauszufinden, ob ein Ansatz dieses Vergleichsmerkmal erfüllt, werden nachfolgende Kontrollbeispiele, angewendet auf die XML Bäume in Abbildung 4.9 und 4.10, verwendet:

Kontrollbeispiel 5.1 - XML Schlüssel:
Ein Vortragender wird identifiziert durch seinen Vornamen und Nachnamen. Der Schlüs-

sel lautet somit: σ = (uni.vortragende.vt, {@vn, @nn}). Angewendet auf den XML Baum in Abbildung 4.9 ergeben sich folgende Werte:

VT	@vn	@nn
1	Laura	Leitner
2	Felix	Leitner

Da die beiden Vortragenden mit der Kombination ihres Vor- und Nachnamens eindeutig identifiziert werden können, ist die Schlüsselbedingung für diesen Baum erfüllt. Würden stattdessen zwei unäre Schlüssel σ = (uni.vortragende.vt, {@vn}) und σ = (uni.vortragende.vt, {@nn}) definiert werden, wäre zwar aufgrund des ersten Constraints eine eindeutige Identifikation möglich (die Werte der beiden Attribute vn unterscheiden sich), der zweite Constraint würde dies jedoch nicht mehr möglich machen, da die Werte der Attribute nn identisch sind. Eine Aufteilung in zwei unäre Constraints würde somit nicht zum gewünschten Ergebnis führen.

Kontrollbeispiel 5.2 - XML Schlüssel:
Die Schlüsselbedingung lautet wiederum σ = (uni.vortragende.vt, {@vn, @nn}) und wird nun für den XML Baum in Abbildung 4.10 verwendet. Dadurch ergeben sich folgende Werte:

VT	@vn	@nn
1	Laura	Leitner
2	Laura	Leitner

Die Vor- und Nachnamen der beiden Vortragenden stimmen überein. Eine eindeutige Identifizierung ist somit nicht möglich und die Schlüsselbedingung für diesen XML Baum nicht erfüllt. Ebenso wäre dies bei der Definition von zwei unären Constraints der Fall, da sich auch hier die Werte wieder gleichen würden und somit die Identifizierung der Vortragenden nicht möglich ist.

Kontrollbeispiel 5.3 - XML FDs:
Das Attribut id eines Vortragenden wird bestimmt durch die Kombination seiner Vor- und Nachnamen. Die Funktionale Abhängigkeit lautet somit: σ = (uni.vortragende.vt, {@vn, @nn} → {@id}). Angewendet auf den XML Baum in Abbildung 4.9 ergeben sich folgende Werte:

@vn	@nn	@id
Laura	Leitner	V1
Felix	Leitner	V2

Da die id-Attribute eindeutig bestimmt werden, ist die Funktionale Abhängigkeit für diesen XML Baum erfüllt. Würden statt einem n-ären Constraint die beiden unären Constraints σ = (uni.vortragende.vt, {@vn} $\rightarrow$ {@id}) und σ = (uni.vortragende.vt, {@nn} $\rightarrow$ {@id}) definiert werden, wäre eine eindeutige Bestimmung der ID mit dem ersten Constraint möglich, nicht jedoch mit dem zweiten.

Kontrollbeispiel 5.4 - XML FDs:
Die Funktionale Abhängigkeit lautet wiederum σ = (uni.vortragende.vt, {@vn, @nn} $\rightarrow$ {@id}). Für den XML Baum in Abbildung 4.10 ergeben sich folgende Werte:

@vn	@nn	@id
Laura	Leitner	V1
Laura	Leitner	V2

Hier ergibt die Kombination von Vor- und Nachnamen der beiden Vortragenden dieselben Werte. Da die Vortragenden aber verschiedene id-Werte haben, ist der Constraint nicht erfüllt. Ebenso nicht erfüllt wären die beiden unären Constraints σ = (uni.vortragende.vt, {@vn} $\rightarrow$ {@id}) und σ = (uni.vortragende.vt, {@nn} $\rightarrow$ {@id}), da auch hier die Werte gleich wären und die IDs nicht eindeutig bestimmt werden könnten.

Kontrollbeispiel 5.5 - XML InclDs:
Der Vor- und Nachname eines Vortragenden einer Lehrveranstaltung (Teilbaum lvas) muss auch in der Liste der Vortragenden (Teilbaum vortragende) vorhanden sein. Die Inklusionsabhängigkeit lautet somit: σ = (uni.lvas.lva.vt, [@vn, @nn]) $\subseteq$ (uni.vortragende.vt, [@vn, @nn]). Für den XML Baum in Abbildung 4.9 ergeben sich dadurch folgende Werte:

Teilbaum uni.lvas.lva.vt

@vn	@nn
Laura	Leitner
–	–

Teilbaum uni.vortragende.vt

@vn	@nn
Laura	Leitner
Felix	Leitner

Da die Kombination des Vor- und Nachnamens der Vortragenden im Teilbaum der Lehrveranstaltungen [Laura, Leitner] auch im Teilbaum der Vortragenden vorhanden ist, ist die Inklusionsabhängigkeit für diesen XML Baum erfüllt. Würden statt einem n-ären Constraint die beiden unären Constraints σ = (uni.lvas. lva.vt, [@vn]) $\subseteq$ (uni.vortragende.vt, [@vn]) und σ = (uni.lvas.lva.vt, [@nn]) $\subseteq$ (uni. vortragende.vt, [@nn]) definiert werden, wären auch diese erfüllt, da die Werte [Laura] und [Leitner] des Teilbaums uni.lvas.lva.vt auch im Teilbaum uni. vortragende.vt vorhanden sind.

Kontrollbeispiel 5.6 - XML InclDs:
Die Inklusionsabhängigkeit lautet wiederum σ = (uni.lvas.lva.vt, [@vn, @nn]) $\subseteq$ (uni.vortragende.vt, [@vn, @nn]) und wird nun für den XML Baum in Abbildung 4.10 verwendet. Die Wertekombination [Felix, Schnell] im Teilbaum lvas ist im Teilbaum vortragende nicht vorhanden und die Inklusionsabhängigkeit ist nicht erfüllt:

Teilbaum uni.lvas.lva.vt

@vn	@nn
Felix	Leitner
–	–

Teilbaum uni.vortragende.vt

@vn	@nn
Laura	Leitner
Laura	Leitner

Würden statt dem n-ären Constraint die beiden unären Constraints σ = (uni.lvas. lva.vt, [@vn]) $\subseteq$ (uni.vortragende.vt, [@vn]) und σ = (uni.lvas.lva.vt, [@nn]) $\subseteq$ (uni.vortragende.vt, [@nn]) verwendet werden, wäre zumindest die zweite Funktionale Abhängigkeit erfüllt, da der Wert [Leitner] sowohl im Teilbaum lvas als auch im Teilbaum vortragende vorhanden ist. Dies würde jedoch nicht die Wirklichkeit spiegeln, da es sich offensichtlich um zwei verschiedene Vortragende handelt.

Zu unterscheiden ist der n-äre Wertvergleich vom baumbasierten Wertvergleich, welcher im nächsten Abschnitt behandelt wird. Konkrete Beispiele, die die Unterschiede verdeutlichen, werden ebenfalls im Abschnitt 4.1.5 angeführt.

4.1.5 Wertvergleich

Für die Verwendung von Integritätsbedingungen ist es notwendig, genau zu definieren unter welcher Bedingung zwei Knoten als "gleich" behandelt werden. [20]

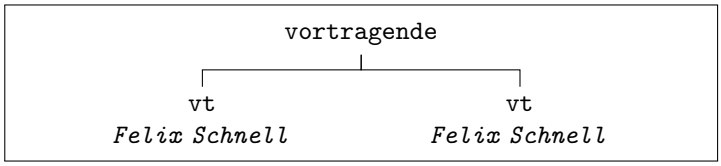

Abbildung 4.11: XML Baum zur Darstellung von Knoten- und Wertgleichheit

Während in relationalen Datenbanken der reine Wertvergleich ausreicht, da Tupel nicht mehrfach vorkommen dürfen, wird in XML zwischen Knoten- und Wertgleichheit unterschieden [55]. Die Knotengleichheit basiert auf der Identität der Knoten. Wenn zwei Knoten n_1 und n_2 dieselben Knoten sind (d.h. dieselbe Identität haben), sind sie gleich (meist geschrieben als $n_1 = n_2$). Bei der Wertgleichheit werden – wie der Name schon sagt – die Werte der Knoten verglichen. Zwei Knoten n_1 und n_2 sind somit gleich, wenn sie die gleichen Werte haben (meist geschrieben als $n_1 =_v n_2$). Hierbei ist zu beachten, dass auch verschiedene Knoten die gleichen Werte haben können. [35]

Für die Definition von Wertgleichheit können verschiedene Notationen verwendet werden. Die einfachste Form der Wertgleichheit basiert auf dem Vergleich von Attribut- oder Textwerten. Zwei Knoten sind somit gleich, wenn sie dieselben Stringwerte (d.h. die gleichen Attribut- bzw. Textwerte) haben. [20]

Die Unterschiede zwischen Knotenvergleich und einfachem Wertvergleich zeigt folgendes Beispiel:

Beispiel 4.3: *Der XML Baum in Abbildung 4.11 zeigt zwei Elemente* `vt` *mit einem Textwert "Felix Schnell". Wird zum Vergleich dieser Elemente ein Knotenvergleich verwendet, sind diese nicht gleich. Wird hingegen ein Wertvergleich verwendet, sind beide Knoten gleich, da beide den Wert "Felix Schnell" haben.*

Eine weitere Möglichkeit ist der Wertvergleich basierend auf der Verkettung von Textwerten, d.h. der Wert eines Elementknotens ist die Verkettung aller direkt und indirekt nachfolgenden Textwerte. Voraussetzung dafür ist, dass ein Baummodell verwendet wird, das gemischten Inhalt in XML Dokumenten unterstützt (siehe Vergleichsmerkmal 4). [53]

Wie sich diese beiden Formen des Wertvergleichs unterscheiden, zeigt folgendes Beispiel:

Beispiel 4.4: *Der XML Baum in Abbildung 4.12 zeigt die Namen zweier Vortragende, welche im Element* `name` *gespeichert sind. Das Element beinhaltet Unterelemente und Textwerte. Wird zum Vergleich der beiden Elemente einfacher Wertvergleich verwendet,*

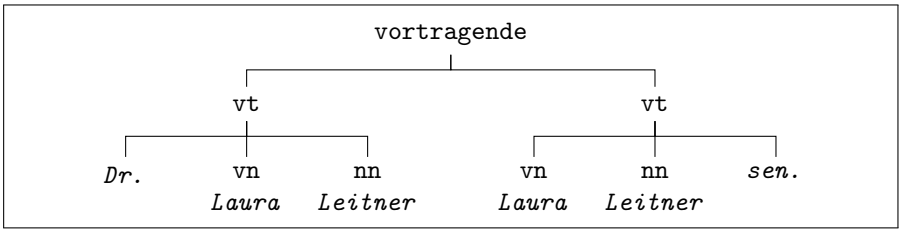

Abbildung 4.12: XML Baum zur Gegenüberstellung von einfachem Wertvergleich und Wertvergleich basierend auf der Verkettung aller direkt und indirekt nachfolgenden Textwerte

müssten die Knoten- und Textwerte einzeln miteinander verglichen werden, d.h. der Vorname der ersten Vortragenden mit dem Vornamen der zweiten Vortragenden, der Nachname der ersten Vortragenden mit dem der zweiten Vortragenden usw. Dies ist eine sehr aufwändige Form des Wertvergleichs, v.a. wenn sich sehr viele Unterelemente im zu vergleichendem Element befinden. Von großer Bedeutung sind in diesem Beispiel auch die Textwerte "Dr." und "sen.", welche sich jeweils nur in einem der Elemente `name` *befinden. Diese würden bei einfachem Textvergleich nicht mit einbezogen werden. Da diese Werte jedoch ausschlaggebend zur Unterscheidung der beiden Elemente* `name` *sind (beim einfachen Textvergleich der Werte für* `vn` *und* `nn` *wären die beiden Namen identisch), sollten sie unbedingt berücksichtigt werden. Wird nun ein Wertvergleich basierend auf der Verkettung aller direkt und indirekt nachfolgenden Textwerte angewendet, ergeben sich für die Elemente* `name` *die Werte "Dr. Laura Leitner" und "Laura Leitner sen.". Eine der Wirklichkeit entsprechende Unterscheidung der beiden Elemente ist somit möglich.*

Beispiele wie das eben angeführte zeigen Anwendungsbereiche, bei denen ein Wertvergleich basierend auf der Verkettung von Textwerten sinnvoll ist. Eine weitere Form ist der baumbasierte Wertvergleich. Dabei wird sowohl die Struktur der Unterknoten (also deren Position, Anzahl, Reihenfolge usw.) als auch die Werte der Knoten miteinbezogen. Zwei Elementknoten sind somit gleich, wenn sie dieselben Stringwerte (also Attribut- oder Textwerte) haben und die Struktur der Unterbäume äquivalent ist. [18]

Baumbasierter Wertvergleich und Wertvergleich basierend auf der Verkettung von Textwerten können zu unterschiedlichen Ergebnissen führen, wie nachfolgende Beispiele zeigen. Während für das Beispiel 4.5 ein Wertvergleich basierend auf der Verkettung von Textwerten besser geeignet ist, ist für das Beispiel 4.6 ein baumbasierter Wertvergleich sinnvoller.

Beispiel 4.5: *Der XML Baum in Abbildung 4.13 zeigt die Geburtsdaten zweier Vortra-*

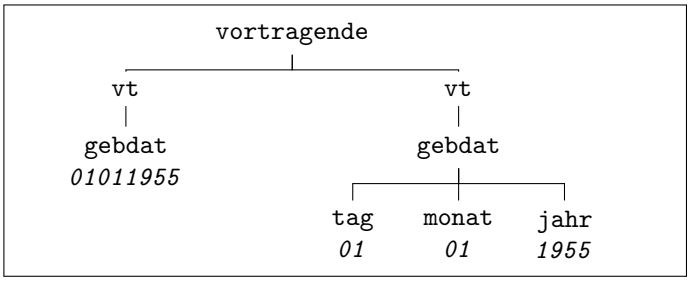

Abbildung 4.13: XML Baum zur Gegenüberstellung vom Wertvergleich basierend auf der Verkettung von Textwerten und baumbasiertem Wertvergleich

```
<wegbeschreibung>
    <anfahrt @richtung = "Linz">
      Bitte folgen Sie dem Straßenverlauf für
        <km> 5 </km>
        km und biegen Sie danach rechts ab.
    </anfahrt>
    <anfahrt @richtung = "Salzburg">
      Bitte folgen Sie dem Straßenverlauf für 5 km
          und biegen Sie danach rechts ab.
    </anfahrt>
</wegbeschreibung >
```

Abbildung 4.14: XML Dokument zur Gegenüberstellung vom Wertvergleich basierend auf der Verkettung von Textwerten und baumbasiertem Wertvergleich

gender. Das Datum des ersten Vortragenden ist in Form des Textwertes "01011955" im Element `gebdat` *gespeichert, während es beim zweiten Vortragenden in die Textwerte der drei Unterelemente* `tag`, `monat` *und* `jahr` *verteilt ist. Würde hier der Wertvergleich basierend auf der Verkettung der Textwerte erfolgen, würden die Werte des Elements* `gebdat` *für beide Vortragende gleich sein. Wird hingegen ein baumbasierter Wertvergleich verwendet, unterscheiden sich die Elemente, da auch die Baumstruktur in den Vergleich mit einbezogen wird. In diesem Beispiel wäre der Wertvergleich mittels Verkettung von Textwerten vorzuziehen, da es sich offensichtlich um das gleiche Geburtsdatum handelt.*

Beispiel 4.6: *Das XML Dokument in Abbildung 4.14 zeigt zwei Wegbeschreibungen, welche als Elemente* `anfahrt` *gespeichert sind. Bei einem Stringvergleich basierend auf der Verkettung aller direkt und indirekt nachfolgenden Textwerte sind die beiden Elemente* `anfahrt` *gleich, da der Inhalt beider Elemente lautet: "Bitte folgen Sie dem Straßenver-*

lauf für 5 km und biegen Sie danach rechts ab.". Wird jedoch ein baumbasierter Wertvergleich verwendet, unterscheiden sich die beiden Elemente, da sie sich sowohl in der Struktur als auch in den Werten unterscheiden (beim baumbasierten Wertvergleich wird auch das Attribut `richtung` *mit in den Vergleich aufgenommen). Da es sich in diesem Beispiel offensichtlich um zwei verschiedene Anfahrtsbeschreibungen handelt (einmal Richtung Linz und einmal Richtung Salzburg), sollten die beiden Elemente* `anfahrt` *als nicht gleich behandelt werden und somit baumbasierter Wertvergleich angewendet werden.*

Die eben gezeigten Beispiele zeigen, dass sowohl der Wertvergleich basierend auf der Verkettung nachfolgender Textwerte als auch der baumbasierte Wertvergleich von Bedeutung sind. Aus diesem Grund werden beide Formen in den Vergleich verschiedener IC-Ansätze mit aufgenommen. Daraus ergeben sich nachfolgende Merkmale 6 und 7.

Vergleichsmerkmal 6: *Der Ansatz ermöglicht die Definition von Integritätsbedingungen mit Wertvergleich von gemischtem Inhalt.*

Vergleichsmerkmal 7: *Der Ansatz ermöglicht die Definition von Integritätsbedingungen mit baumbasiertem Wertvergleich*

Um herauszufinden, ob das Vergleichsmerkmal 6 von einem IC-Ansatz erfüllt wird, werden die Kontrollbeispiele 6.1 bis 6.6 verwendet.

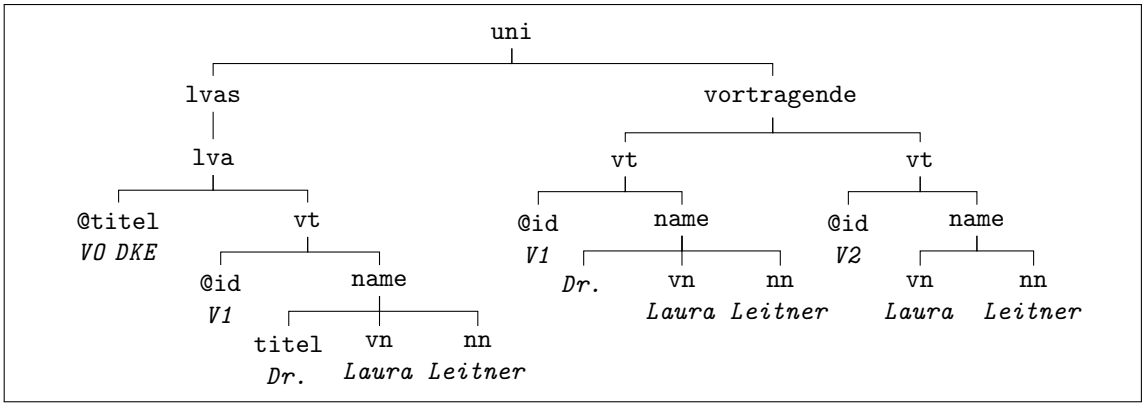

Abbildung 4.15: XML Baum 1 zur Überprüfung von gemischtem Inhalt in XML Constraints

Kontrollbeispiel 6.1 - XML Schlüssel:
Ein Vortragender wird durch seinen Namen identifiziert. Der Schlüssel lautet somit:

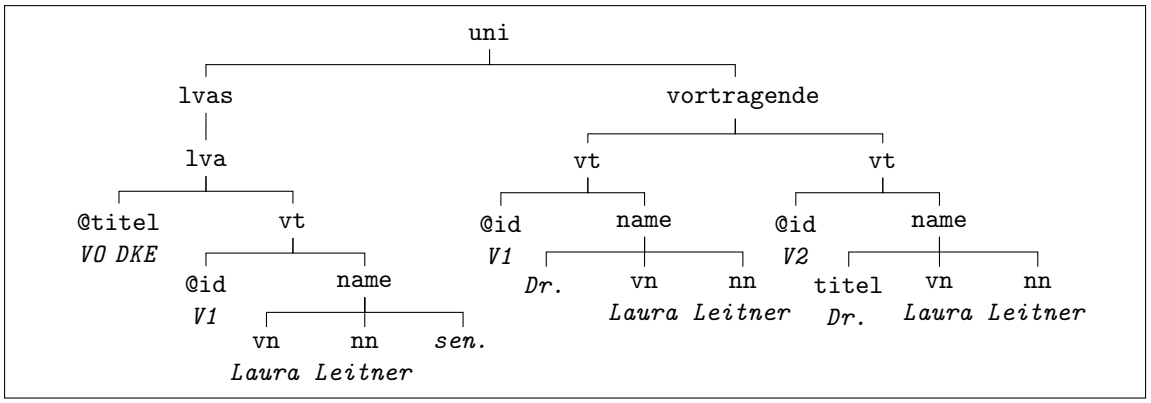

Abbildung 4.16: XML Baum 2 zur Überprüfung von gemischtem Inhalt in XML Constraints

σ = (uni.vortragende.vt, {name}). Bei einem Wertvergleich von gemischtem Inhalt ist der Wert des Elements `name` *die Verkettung sämtlicher direkt und indirekt nachfolgender Textwerte. Angewendet auf den XML Baum in Abbildung 4.15 ergeben sich folgende Werte:*

VT	*name*
1	*Dr. Laura Leitner*
2	*Laura Leitner*

Da sich diese Werte unterscheiden, ist eine eindeutige Identifikation der Vortragenden möglich. Die Schlüsselbedingung ist somit erfüllt. Hier würde auch ein baumbasierter Wertvergleich zum gewünschten Ergebnis führen, da sich, neben der Baumstruktur, auch die zu vergleichenden Werte unterscheiden. Dies verdeutlicht auch, dass die Werte zweier Elemente beim baumbasierten Wertvergleich automatisch verschieden sind, sobald sie dies auch beim Wertvergleich von gemischtem Inhalt der Element sind.

Kontrollbeispiel 6.2 - XML Schlüssel:
Der Schlüssel lautet wiederum σ = (uni.vortragende.vt, {name}). Im XML Baum in Abbildung 4.16 wird diese Bedingung verletzt, da die `name`*-Werte der beiden Vortragenden gleich sind und sie somit nicht eindeutig identifiziert werden können:*

VT	*name*
1	*Dr. Laura Leitner*
2	*Dr. Laura Leitner*

Die Schlüsselbedingung ist daher nicht erfüllt. Würde stattdessen jedoch ein baumbasierter Wertvergleich verwendet werden, wäre der Constraint erfüllt, da sich die Unterbäume der `name`*-Elemente der beiden Vortragenden im Teilbaum* `vortragende` *unterscheiden und die Vortragenden somit eindeutig identifiziert werden könnten. Dies würde allerdings nicht dem für dieses Kontrollbeispiel gewünschtem Ergebnis entsprechen.*

Kontrollbeispiel 6.3 - XML FDs:
Das `id`*-Attribut eines Vortragenden wird bestimmt durch seinen Namen. Die Funktionale Abhängigkeit lautet somit:* σ `= (uni.vortragende.vt, {name}` $\rightarrow$ `{@id})`*. Für den XML Baum in Abbildung 4.15 ergeben sich folgende Werte:*

`name`	`@id`
Dr. Laura Leitner	*V1*
Laura Leitner	*V2*

Da die `id`*-Attribute eindeutig bestimmt werden können, ist die Funktionale Abhängigkeit erfüllt. Auch hier würde ein baumbasierter Wertvergleich zum selben Ergebnis führen, da eine Funktionale Abhängigkeit, die bereits mit Wertvergleich von gemischten Inhalt erfüllt ist, automatisch auch mit baumbasierten Wertvergleich erfüllt ist.*

Kontrollbeispiel 6.4 - XML FDs:
Die Funktionale Abhängigkeit lautet wiederum σ `= (uni.vortragende.vt, {name}` $\rightarrow$ `{@id})`*. Angewendet auf den XML Baum in Abbildung 4.16 ergeben sich folgende Werte:*

`name`	`@id`
Dr. Laura Leitner	*V1*
Dr. Laura Leitner	*V2*

Da beide Vortragende denselben Namen jedoch verschiedene `id`*-Werte haben, ist die Funktionale Abhängigkeit nicht erfüllt. Würde hingegen ein baumbasierter Wertvergleich verwendet werden, wäre der Constraint erfüllt, da sich die Unterbäume der beiden Elemente* `name` *unterscheiden und die IDs eindeutig bestimmt werden könnten. Dieses Ergebnis ist für dieses Kontrollbeispiel jedoch nicht gefordert.*

Kontrollbeispiel 6.5 - XML InclDs:
Der Name eines Vortragenden einer Lehrveranstaltung muss auch in der Liste der Vortragenden vorhanden sein. Die Inklusionsabhängigkeit lautet somit: σ `= (uni.lvas.lva.`

vt, [name]) ⊆ (uni.vortragende.vt, [name]). Angewendet auf den XML Baum in Abbildung 4.15 ergeben sich folgende Werte:

Teilbaum lvas.lva.vt	*Teilbaum vortragende.vt*
name	*name*
Dr. Laura Leitner *–*	*Dr. Laura Leitner* *Laura Leitner*

Da der Wert des Elements name der Vortragenden im Teilbaum der Lehrveranstaltungen auch im Teilbaum der Vortragenden gespeichert ist, ist diese Inklusionsabhängigkeit erfüllt. Würde für diesen Constraint jedoch ein baumbasierter Wertvergleich verwendet werden, wäre dieser nicht erfüllt, da der Unterbaum des name-Elements des Teilbaums lvas nicht im Teilbaum vortragende vorhanden ist. Dies entspricht aber nicht dem gewünschten Ergebnis.

Kontrollbeispiel 6.6 - XML InclDs:
Die Inklusionsabhängigkeit lautet wiederum σ = (uni.lvas.lva.vt, [name]) ⊆ (uni.vortragende.vt, [name]) und wird nun für den XML Baum in Abbildung 4.16 verwendet. Daraus ergeben sich folgende Werte:

Teilbaum lvas.lva.vt	*Teilbaum vortragende.vt*
name	*name*
Laura Leitner sen. *–*	*Dr. Laura Leitner* *Dr. Laura Leitner*

Der Name der Vortragenden im Teilbaum lvas ist im Teilbaum vortragende nicht vorhanden ist, sodass die Inklusionsabhängigkeit nicht erfüllt ist. Zu demselben Ergebnis würde auch ein baumbasierter Wertvergleich führen, da sich bereits die Textwerte unterscheiden und somit auch die Unterbäume der Elemente nicht gleich sein können.

Als nächstes wird auf das Vergleichsmerkmal 7 eingegangen, dass den baumbasierten Wertvergleich betrifft. Dafür werden die Kontrollbeispiele 7.1 - 7.6 verwendet. Diese werden anhand nachfolgender XML Bäume geprüft. Die Bäume wurden aus Platzgründen jeweils in zwei Teilbäume unterteilt, sodass der Teilbaum in Abbildung 4.17 gemeinsam mit dem Teilbaum in Abbildung 4.18 einen XML Baum bildet, ebenso wie die Teilbäume in Abbildung 4.19 und 4.20 zusammen den zweiten XML Baum bilden.

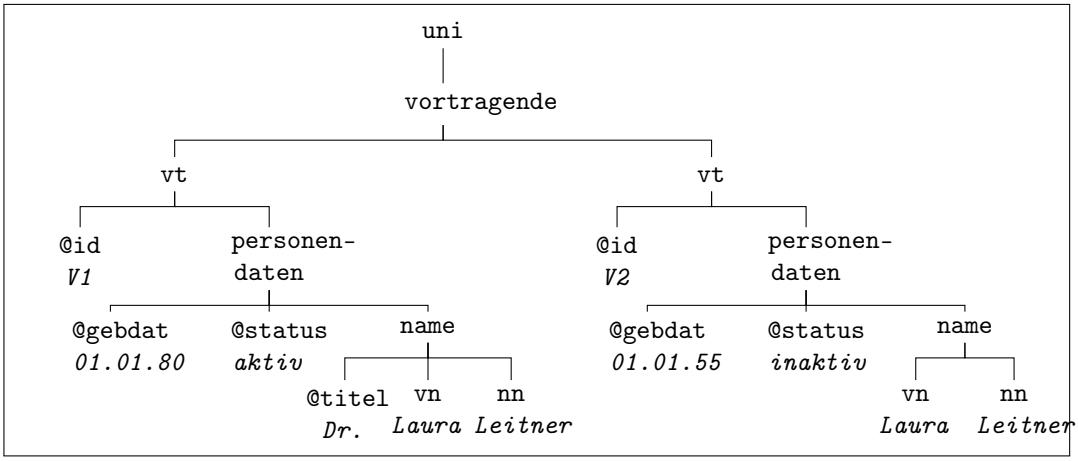

Abbildung 4.17: XML Baum 1 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Vortragende

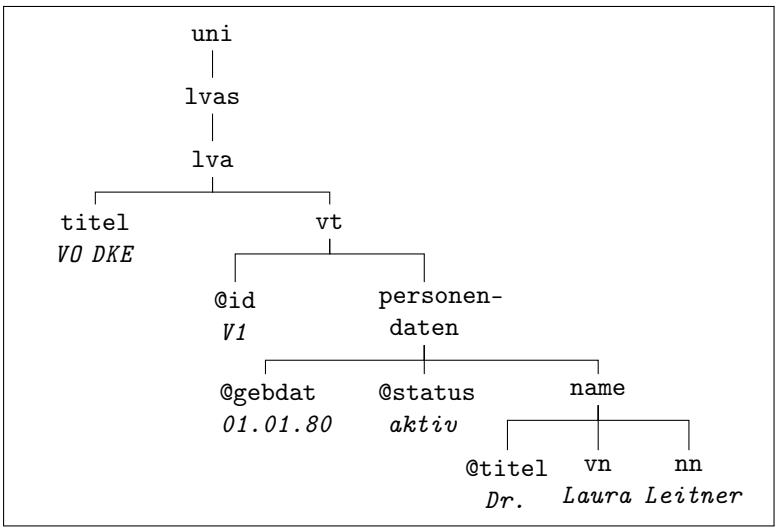

Abbildung 4.18: XML Baum 1 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Lehrveranstaltungen

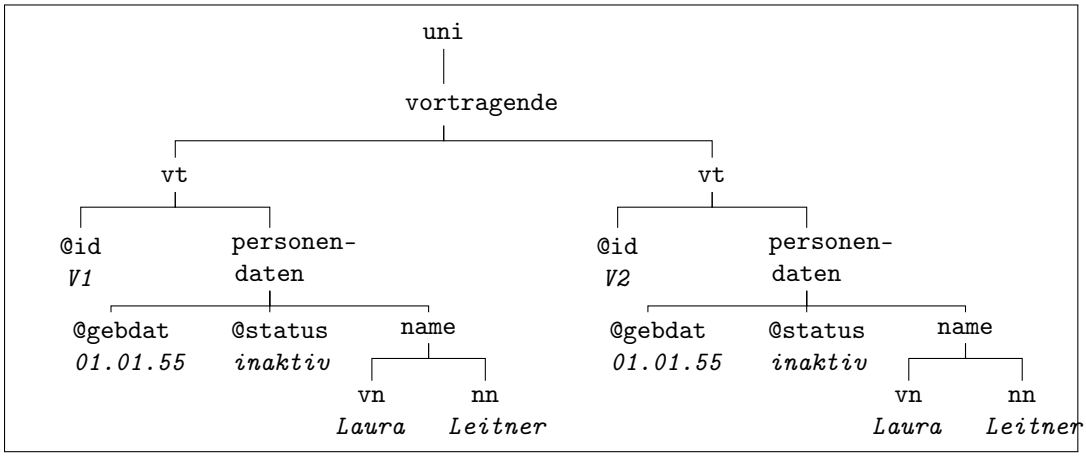

Abbildung 4.19: XML Baum 2 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Vortragende

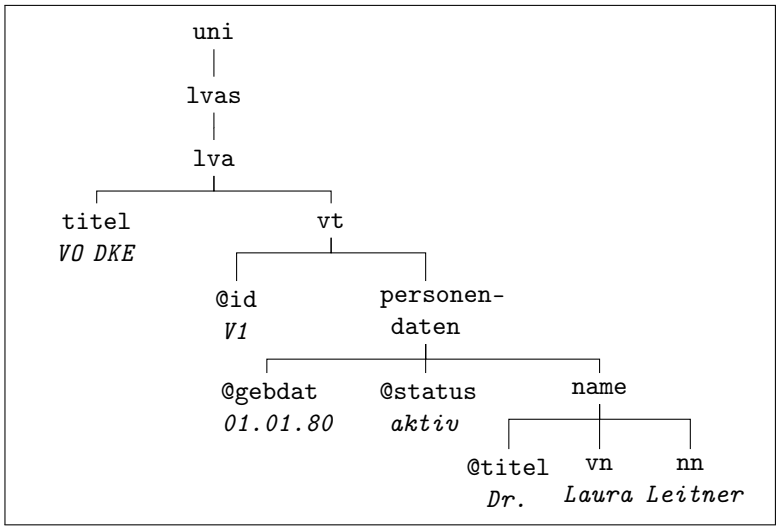

Abbildung 4.20: XML Baum 2 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Lehrveranstaltungen

Kontrollbeispiel 7.1 - XML Schlüssel:
Ein Vortragender wird mit seinen Personendaten (Element `personendaten`*) identifiziert. Der Schlüssel lautet* σ *=* `(uni.vortragende.vt, {personendaten})`*. Die Gleichheit zweier Knoten soll hierbei auf dem Vergleich ihrer Unterbäume basieren. Im XML Baum in Abbildung 4.17 unterscheiden sich die beiden Elemente* `personendaten` *sowohl durch ihre Baumstruktur (bei der zweiten Vortragenden fehlt das Attribut* `titel`*), als auch bei den Werten der Attribute* `gebdat` *und* `status`*. Eine eindeutige Identifizierung der Vortragenden ist somit möglich und der Constraint ist erfüllt. Würde statt einem baumbasierten*

Wertvergleich ein Wertvergleich basierend auf der Verkettung von Textwerten verwendet werden, wäre der Constraint nicht erfüllt, da die zum Vergleich herangezogenen Werte der Elemente `vn` *und* `nn` *identisch sind und somit eine eindeutige Identifikation der Vortragenden nicht möglich ist. Da es sich offensichtlich um zwei verschiedene Vortragende handelt, wäre dieses Ergebnis falsch.*

Kontrollbeispiel 7.2 - XML Schlüssel:
Der Schlüssel lautet wiederum σ `= (uni.vortragende.vt, {personendaten})`*, wird jedoch nun für den XML Baum in Abbildung 4.19 verwendet. Hier ist der Constraint nicht erfüllt, weil die Elementknoten* `personendaten` *der Vortragenden sowohl die gleiche Baumstruktur als auch die gleichen Werte haben und somit nicht eindeutig identifiziert werden können. Beim Wertvergleich basierend auf der Verkettung von Textwerten wäre der Schlüssel ebenfalls nicht erfüllt.*

Kontrollbeispiel 7.3 - XML FDs:
Die ID eines Vortragenden wird durch seine Personendaten bestimmt. Die dazugehörige Funktionale Abhängigkeit lautet: σ `= (uni.vortragende.vt, {personendaten}` $\rightarrow$ `{@id})`*. Der XML Baum in Abbildung 4.17 erfüllt diese Bedingung, da sich sowohl die Baumstruktur der beiden Elemente* `personendaten` *durch das fehlende Attribut* `titel` *der zweiten Vortragenden als auch deren Werte unterscheiden. Die ID kann somit bestimmt werden und der Constraint ist erfüllt. Würde hingegen ein Wertvergleich basierend auf der Verkettung von Textwerten verwendet werden, wäre der Constraint nicht erfüllt, da die Textwerte der Elemente* `personendaten` *gleich sind. Dies würde jedoch nicht der Realität entsprechen und zu einem falschen Ergebnis führen.*

Kontrollbeispiel 7.4 - XML FDs:
Die Funktionale Abhängigkeit lautet nochmals σ `= (uni.vortragende.vt, {personendaten}` $\rightarrow$ `{@id})`*. Angewendet auf den XML Baum in Abbildung 4.19 ist der Constraint nicht erfüllt, da sowohl die Baumstruktur als auch die Werte des Elements* `personendaten` *gleich sind und die ID nicht eindeutig bestimmt werden kann. Zu demselben Ergebnis würde ein Wertvergleich basierend auf der Verkettung von Textwerten führen.*

Kontrollbeispiel 7.5 - XML InclDs:
Die Personendaten eines Vortragenden im Teilbaum `lvas` *muss gleichzeitig im Teilbaum* `vortragende` *vorhanden sein. Die Inklusionsabhängigkeit lautet:*σ `= (uni.lvas.lva.vt, [personendaten])` $\subseteq$ `(uni.vortragende.vt, [personendaten])`*. Angewendet am XML Baum in den Abbildungen 4.17 und 4.18 ist dieser Constraint erfüllt, da die*

Personendaten sowie die Baumstruktur der Vortragenden im Teilbaum `lva` *gleich sind der der zweiten Vortragenden im Teilbaum* `vortragende`*.*

Kontrollbeispiel 7.6 - XML InclDs:
Wiederum lautet die Inklusionsabhängigkeit σ = `(uni.lvas.lva.vt, [personendaten])` $\subseteq$ `(uni.vortragende.vt, [personendaten])`*. Wird sie für den XML Baum in den Abbildungen 4.19 und 4.20 verwendet, ist die Bedingung nicht erfüllt, da weder die Werte noch die Baumstruktur des Elements* `personendaten` *der Vortragenden im Teilbaum* `lvas` *mit denen im Teilbaum* `vortragende` *übereinstimmen. Würde jedoch ein Wertvergleich basierend auf der Verkettung von Textwerten verwendet werden, wäre der Constraint erfüllt, weil lediglich die Werte der Elemente* `vn` *und* `nn` *zum Vergleich herangezogen werden würden und diese identisch sind.*

Neben der Unterscheidung zwischen einem Wertvergleich basierend auf der Verkettung von Textwerten und einem baumbasierten Wertvergleich, ist auch eine Abgrenzung zu n-ären Constraints von Bedeutung. Das folgende Beispiel soll dies zeigen:

Beispiel 4.7: *Der XML Baum in Abbildung 4.21 zeigt die Daten zweier Vortragender, wobei ein Wertvergleich des Elements* `name` *vorgesehen ist. Wird dafür ein Wertvergleich basierend auf der Verkettung von Textwerten verwendet, sind die beiden Elemente verschieden, da sich die Werte "Laura Leitner" und "Dr. Laura Leitner" der beiden* `name`*-Elemente voneinander unterscheiden. Wird hierfür ein baumbasierter Wertvergleich verwendet, führt dies zu dem gleichen Ergebnis, weil sich sowohl die Werte der einzelnen Knoten als auch die Baumstruktur unterscheiden. Würde hingegen ein n-ärer Wertvergleich verwendet werden, könnten lediglich die Werte der Elemente* `vn` *und* `nn` *zum Vergleich herangezogen werden. Da diese Werte für beide Vortragende gleich sind, können sie nicht voneinander unterschieden werden.*

Der Wertvergleich basierend auf der Verkettung von Textwerten bzw. der baumbasierte Wertvergleich und n-ärer Wertvergleich schließen sich jedoch nicht aus, sondern können auch gemeinsam verwendet werden, wie folgendes Beispiel zeigt:

Beispiel 4.8: *Für den XML Baum in Abbildung 4.21 soll der Wert des Elements* `vt` *definiert werden, der aus den Unterelementen* `name` *und* `inst` *besteht. Hierfür wäre es beispielsweise möglich, den n-ären Wertvergleich mit baumbasiertem Wertvergleich oder dem Wertvergleich basierend auf der Verkettung von Textwerten zu verbinden. Der Wert eines* `vt`*-Elements würde sich dann aus den Unterelementen* `name` *und* `inst` *zusammen-*

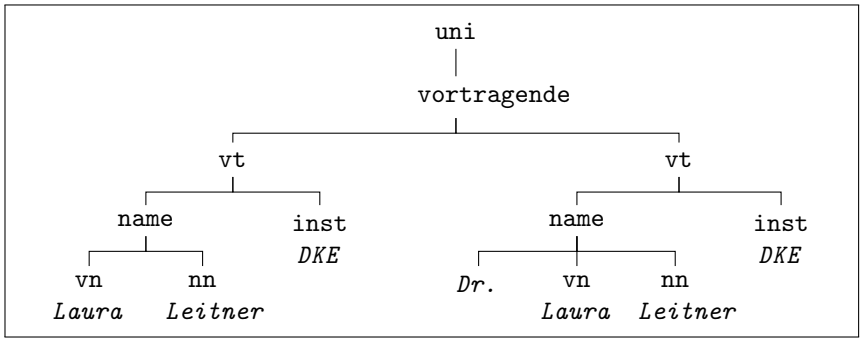

Abbildung 4.21: XML Baum zum Unterscheiden zwischen Wertvergleich durch Verkettung von Textwerten sowie baumbasiertem Wertvergleich und n-ärem Wertvergleich

setzen, wobei für `name` *ein baumbasierter Wertvergleich oder ein Wertvergleich basierend auf der Verkettung von Textwerten verwendet werden kann.*

Neben den soeben beschriebenen Formen des Wertvergleichs können noch weitere Eigenschaften beim Vergleich von Knoten mit einbezogen werden. Eine Möglichkeit ist, auch den Typ eines Wertes zu beachten. Ein Beispiel dafür ist der Vergleich der beiden Werte "+1" und "1". Wird ein einfacher Stringvergleich verwendet, sind diese beiden Werte verschieden. Wird jedoch der Datentyp `integer` für beide Werte definiert und zum Vergleich verwendet, sind diese beiden Werte gleich. Eine weitere Vorgehensweise wäre ein verstärktes Einbeziehen des Benutzers um dadurch einen benutzerdefinierten Wertvergleich zu ermöglichen. [20] Obwohl diese Formen des Wertvergleichs die Ausdruckskraft von Constraint erweitern könnten, wurden sie nur in wenigen IC-Ansätzen integriert. Aus diesem Grund werden diese Möglichkeiten nicht in den Vergleich der Ansätze im Zuge dieser Diplomarbeit integriert.

4.1.6 Geltungsbereich von Constraints

Daten der realen Welt können meist in verschiedene Bereiche unterteilt bzw. bestimmten Aspekten zugeteilt werden. Werden diese Daten mit Hilfe einer relationalen Datenbank gespeichert, kann dies mittels Speicherung in verschiedenen Relationen abgebildet werden. Es wird also für jeden Bereich eine eigene Relation verwendet, innerhalb dieser nur die diesem Bereich zugehörigen Daten gespeichert werden. [3]

Werden die Daten hingegen in Form eines XML Dokuments gespeichert, ist eine solche Unterteilung auch unter Zuhilfenahme der hierarchischen Struktur der XML Dokumente möglich. Ein XML Baum beinhaltet einen Wurzelknoten und eine beliebige Anzahl

an Unterbäumen. Diese Unterbäume können zur Abgrenzung der Daten in verschiedene Teilbereiche verwendet werden. [33]

Diese Eigenschaft ist auch für die Definition von XML Integritätsbedingungen von Bedeutung. Es gibt Constraints die im gesamten XML Dokument gültig sind. Diese werden absolute Constraints genannt. Wird jedoch der Geltungsbereich (engl.: Scope) eines Constraints auf einen bestimmten Bereich beschränkt, werden sie als relative Constraints bezeichnet. Sie gelten dann nur in den Unterbäumen einer genau definierten Menge von Elementknoten und haben keine Bedeutung für das restliche XML Dokument. [33]

Im weiteren Verlauf dieser Diplomarbeit wird eine an Buneman [18] angelehnte Syntax zur Definition von relativen Constraints verwendet. Hierbei definiert ein *Context Path* jenen Bereich, in dem der Constraint gelten soll. Eine relative Funktionale Abhängigkeit wird demnach definiert als $\sigma = (C, (S, \{F_1, \cdots, F_n\} \rightarrow \{F'_1, \cdots, F'_n\}))$. Die Funktionale Abhängigkeit $\sigma = (S, \{F_1, \cdots, F_n\} \rightarrow \{F'_1, \cdots, F'_n\})$ gilt somit nur in den mit dem *Context Path C* zu erreichenden Teilbäumen. Schlüssel und Fremdschlüssel werden definiert als $\sigma = (C, (S, \{F_1, \cdots, F_n\}))$. Auch hier bezeichnet *C* den Geltungsbereich des Schlüssels $\sigma = (S, \{F_1, \cdots, F_n\})$. Relative Inklusionsabhängigkeiten werden folglich definiert als $\sigma = (C, (S, [F_1, \cdots, F_n]) \subseteq (S', [F'_1, \cdots, F'_n]))$, wiederum mit *C* zum Eingrenzen des Geltungsbereichs der Inklusionsabhängigkeit $\sigma = (S, [F_1, \cdots, F_n]) \subseteq (S', [F'_1, \cdots, F'_n])$.

Mit Hilfe von relativen Constraints können auch Bedingungen abgebildet werden, die mit absoluten Constraints nicht definiert werden könnten. Sie umfassen somit eine viel größere Bandbreite an Anwendungsmöglichkeiten. [18] Das folgende Beispiel soll dies verdeutlichen:

Beispiel 4.9: *In einem XML Dokument werden alle Lehrveranstaltungen einer Universität in Form eines Elementknotens* `lva` *abgebildet. In jedem dieser Elemente werden die für diese LVA eingeschriebenen Studenten als Unterelemente* `student` *gespeichert. Des Weiteren beinhaltet jedes* `student`*-Element ein Attribut* `name`*, welches den Studenten innerhalb einer LVA eindeutig identifiziert. Dementsprechend lautet der relative Schlüssel* $\sigma =$ `(uni.lvas.lva, (student, {@name}))`*. Würde statt diesem relativen ein absoluter Schlüssel* $\sigma =$ `(uni.lvas.lva.student, {@name})` *definiert werden, wäre es nicht erlaubt, dass ein Student gleichzeitig für mehrere Lehrveranstaltungen angemeldet ist.*

Das nächste Vergleichsmerkmal lautet somit:

Vergleichsmerkmal 8: *Der Ansatz ermöglicht die Definition von relativen Integritätsbe-*

dingungen.

Dieses Merkmal soll anhand nachfolgender Kontrollbeispiele und den zugehörigen XML Bäumen in den Abbildungen 4.22 und 4.23 geprüft werden.

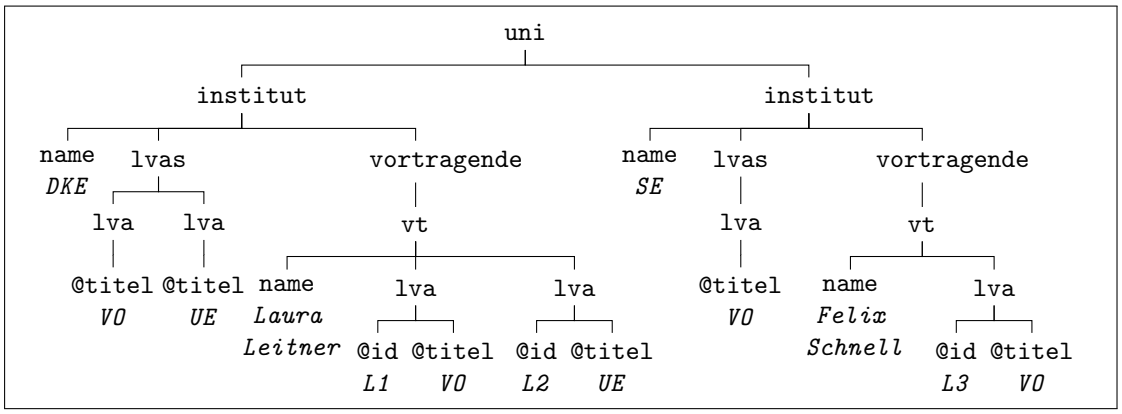

Abbildung 4.22: XML Baum 1 zur Überprüfung des Geltungsbereichs

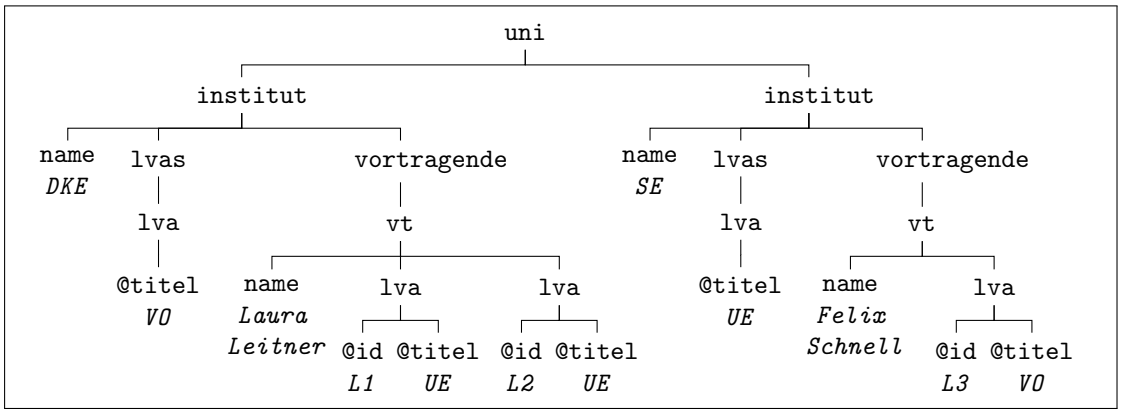

Abbildung 4.23: XML Baum 2 zur Überprüfung des Geltungsbereichs

Kontrollbeispiel 8.1 - XML Schlüssel:
Eine Lehrveranstaltung wird innerhalb eines Vortragenden durch deren Titel identifiziert. Dementsprechend lautet der Schlüssel: σ = `(uni.institut.vortragende.vt, (lva, {@titel}))`. *Für den XML Baum in Abbildung 4.22 ergeben sich dafür folgende Werte:*

Vortragende Laura Leitner

lva	*@titel*
1	*VO*
2	*UE*

Vortragender Felix Schnell

lva	*@titel*
3	*VO*

Die Schlüsselbedingung ist demnach erfüllt. Würde statt einem relativen ein absoluter Schlüssel σ = (uni.institut.vortragende.vt.lva, {@titel}) definiert werden, wäre dieser jedoch nicht erfüllt, da im gesamten XML Baum der Wert VO für das Attribut @titel mehrfach vorhanden ist und die Lehrveranstaltungen somit nicht mehr eindeutig identifiziert werden könnten. Dies würde aber nicht dem gewünschten Ergebnis entsprechen.

Kontrollbeispiel 8.2 - XML Schlüssel:
Der Schlüssel σ = (uni.institut.vortragende.vt, (lva, {@titel})) wird nun für den XML Baum in Abbildung 4.23 verwendet. Hierbei ergeben sich folgende Werte:

Vortragende Laura Leitner

lva	*@titel*
1	*UE*
2	*UE*

Vortragender Felix Schnell

lva	*@titel*
3	*VO*

Da die erste Vortragende in diesem XML Baum zwei Lehrveranstaltungen mit demselben Titel hat, können diese nicht eindeutig identifiziert werden. Die Schlüsselbedingung ist somit nicht erfüllt. Ebenso wäre dies bei der Definition eines absoluten Schlüssels der Fall, da ein Schlüssel, der bereits für einen eingeschränkten Geltungsbereich nicht erfüllt ist, auch nicht für das gesamte XML Dokument erfüllt sein kann.

Kontrollbeispiel 8.3 - XML FDs:
Innerhalb eines Vortragenden wird die ID einer Lehrveranstaltung mit dessen Titel bestimmt. Die Funktionale Abhängigkeit lautet somit: σ = (uni.institut.vortragende.vt, (lva, {@titel} $\rightarrow$ {@id})). Für den XML Baum in Abbildung 4.22 ergeben sich dadurch folgende Werte:

Vortragende Laura Leitner

@titel	@id
VO	L1
UE	L2

Vortragender Felix Schnell

@titel	@id
VO	L3

Da die IDs der Lehrveranstaltungen eines Vortragenden eindeutig bestimmt werden können, ist der Constraint erfüllt. Würde diese Bedingung jedoch als absoluter Constraint σ = (uni.institut.vortragende.vt.lva, {@titel} $\rightarrow$ {@id}) *definiert werden, wäre dieser nicht erfüllt, da die IDs der beiden Lehrveranstaltungen mit dem Titel* VO *nicht identisch sind.*

Kontrollbeispiel 8.4 - XML FDs:
Die Funktionale Abhängigkeit σ = (uni.institut.vortragende.vt, (lva, {@titel} $\rightarrow$ {@id})) *wird nun für den XML Baum in Abbildung 4.23 verwendet. Der Constraint ist hierbei nicht erfüllt, da beide Lehrveranstaltungen der Vortragenden Laura Leitner denselben Titel, jedoch unterschiedliche IDs haben, wie nachfolgende Auflistung zeigt:*

Vortragende Laura Leitner

@titel	@id
UE	L1
UE	L2

Vortragender Felix Schnell

@titel	@id
VO	L3

Ebenfalls nicht erfüllt wäre die absolute Funktionale Abhängigkeit σ = (uni.institut vortragende.vt.lva, {@titel} $\rightarrow$ {@id}), *da, wenn eine Funktionale Abhängigkeit bereits in einem eingeschränkten Bereich nicht gültig ist, sie dies im gesamten XML Dokument auch nicht sein kann.*

Kontrollbeispiel 8.5 - XML InclDs:
Für den XML Baum in Abbildung 4.22 wird die Inklusionsabhängigkeit σ = (uni.institut, (vortragende.vt.lva, [@titel]) $\subseteq$ (lvas.lva, [@titel])) *defi-*

niert. Sie besagt, dass innerhalb eines Instituts der Name einer Lehrveranstaltung im Teilbaum `vortragende` *auch im Teilbaum* `lvas` *vorhanden sein muss. Wird dieser Constraint für den XML Baum in Abbildung 4.22 verwendet, ergeben sich folgende Werte:*

Institut `DKE`

Teilbaum `vortragende`

`vt`	`lva`	`@titel`
Laura Leitner	*1*	*VO*
	2	*UE*

Teilbaum `lvas`

`lva`	`@titel`
1	*VO*
2	*UE*

Institut `SE`

Teilbaum `vortragende`

`vt`	`lva`	`@titel`
Felix Schnell	*1*	*VO*

Teilbaum `lvas`

`lva`	`@titel`
1	*VO*

Diese Auflistung zeigt, dass die Inklusionsabhängigkeit erfüllt ist, da die Namen der LVAs im Teilbaum `vortragende` *auch im Teilbaum* `lvas` *eines Instituts vorhanden sind. Würde statt einer relativen eine absolute Inklusionsabhängigkeit definiert werden, würde diese lauten:* $\sigma =$ `(uni.institut.vortragende.vt.lva, [@titel])` $\subseteq$ `(uni.institut.lvas.lva, [@titel])`*. Für diesen XML Baum wäre auch sie erfüllt, da, wenn eine Inklusionsabhängigkeit bereits in einem Teilbereich des XML Dokuments erfüllt ist, sie automatisch auch im gesamten XML Dokument erfüllt sein muss.*

Kontrollbeispiel 8.6 - XML InclDs:
Die Inklusionsabhängigkeit $\sigma =$ `(uni.institut, (vortragende.vt.lva, [@titel])` $\subseteq$ `(lvas.lva, [@titel]))` *wird nun für den XML Baum in Abbildung 4.23 verwendet. Daraus ergeben sich folgende Werte:*

Institut `DKE`

Teilbaum `vortragende`

`vt`	`lva`	`@titel`
Laura Leitner	*1*	*UE*
	2	*UE*

Teilbaum `lvas`

`lva`	`@titel`
1	*VO*

Institut `SE`

Teilbaum `vortragende`

`vt`	`lva`	`@titel`
Felix Schnell	*1*	*VO*

Teilbaum `lvas`

`lva`	`@titel`
1	*UE*

Der Constraint ist für diesen XML Baum nicht erfüllt, weil die LVAs des Teilbaums `vortragende` *nicht im Teilbaum* `lvas` *desselben Instituts vorhanden sind. Würde diese Inklusionsabhängigkeit jedoch als absoluter Constraint definiert werden, wäre sie erfüllt, da die Titel der Lehrveranstaltungen eines Vortragenden eines Instituts jeweils in der Liste der Lehrveranstaltungen des anderen Instituts vorhanden sind. Dies würde zu einem falschen Ergebnis führen.*

4.1.7 Nicht oder mehrfach vorhandene Datenobjekte

Wie bereits in den Grundlagen in Kapitel 2 erklärt wurde, sind XML Daten semistrukturiert. Eine sehr wichtige Eigenschaft von semistrukturierten Daten ist, dass es keine Anforderungen an die Datenstruktur gibt. Somit ist es auch erlaubt, dass Datenobjekte nicht vorhanden sind. [33] Wird beispielsweise bei einem Vortragenden (Element `vt`) das Geburtsdatum als Element `gebdat` gespeichert, wäre es gemäß der W3C-Spezifikation auch erlaubt, dass bei einzelnen (oder allen) Vortragenden dieses Element nicht vorhanden ist.

Andererseits ist es auch möglich, dass Datenobjekte mehrfach vorhanden sind. [33] Dies wäre beispielsweise der Fall, wenn ein Vortragender (Element `vt`) verschiedene Lehrveranstaltungen abhält und somit mehrere Unterelemente `lva` im Element `vt` vorhanden sind.

Da diese Eigenschaften auch Einfluss auf die Verwendung von Constraints haben, wird im Folgenden etwas näher darauf eingegangen.

Nicht vorhandene Datenobjekte

Sind bestimmte Datenwerte eines XML Dokuments nicht vorhanden, können die Gründe dafür verschieden sein. In relationalen Datenbanksystem werden gemäß Atzeni und De Antonellis [8] zwei Möglichkeiten unterschieden. Diese Unterscheidung kann auch für XML Dokumente verwendet werden.

Die erste Möglichkeit ist, dass die Datenwerte auch in der Realität nicht vorhanden sind (engl.: does not exist, kurz: *dne*) [8]. Dies wäre beispielsweise in einem XML Dokument der Fall, in dem die Daten der Vortragenden in den Unterelementen `name` und `institut` gespeichert werden, für einen Gastvortragenden jedoch kein Wert für das Element `institut` vorhanden ist – weder in der Wirklichkeit noch im XML Dokument.

Ein anderer Grund kann sein, dass die Datenwerte zwar in der Realität existieren, in der

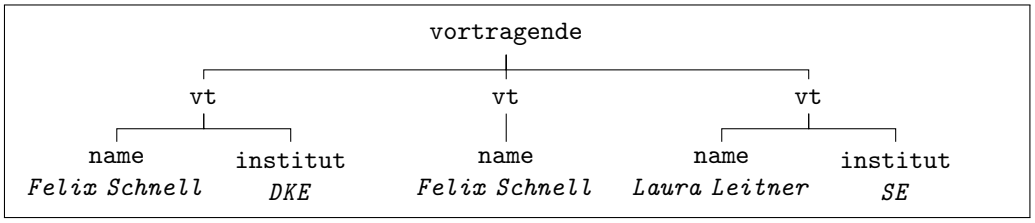

Abbildung 4.24: XML Baum zur Darstellung eines falschen Ergebnisses beim Ignorieren nicht vorhandener Datenobjekte

Datenbank jedoch nicht vorhanden sind (engl.: unknown, kurz: *unk*) [8]. Das wäre beispielsweise der Fall, wenn für einen Vortragenden, der einem Institut angehört, im XML Dokument kein Wert für das Element `institut` vorhanden ist. Dies würde eine falsche Darstellung der Wirklichkeit sein.

Des Weiteren ist es möglich, dass nicht bekannt ist, ob bestimmte Datenwerte in der Realität vorhanden sind oder nicht. Es sind somit keine Informationen über die Abwesenheit der Werte bekannt (engl.: no information, kurz *no info*). [49, 67]

Wichtig ist in diesem Zusammenhang auch, dass nicht vorhandene Datenobjekte eindeutig vom Wert "0" zu unterscheiden sind. [26]

Die Tatsache, dass nicht vorhandene Datenobjekte in XML Dokumenten erlaubt sind, wirken sich auch auf die Verwendung von Constraints aus [55]. Wichtig ist hierbei, dass gewährleistet wird, dass eine semantisch korrekte und der Wirklichkeit entsprechende Umsetzung des Constraints möglich ist. [49]

Es gibt verschiedene Möglichkeiten wie IC-Ansätze mit nicht vorhandenen Datenobjekten umgehen. Eine Möglichkeit besteht darin, Knoten mit fehlenden Werten zu ignorieren. Somit wird die Erfüllbarkeit des Constraints nur für jene Datenobjekte geprüft, die keine fehlenden Datenobjekte beinhalten. [49] Dies kann jedoch zu falschen Ergebnissen führen, wie nachfolgendes Beispiel anhand dem XML Baum in Abbildung 4.24 zeigt:

Beispiel 4.10: *Der XML Baum in Abbildung 4.24 zeigt die Daten dreier Vortragenden, wobei beim zweiten Vortragenden das Element `institut` nicht vorhanden ist. Für einen Constraint werden nun die Werte der Elemente `name` und `institut` zum Vergleich der Elemente `vt` herangezogen. Würde das zweite Element `vt` aufgrund des fehlenden Unterelements `institut` ignoriert werden, würden die verbleibenden Elemente `vt` wertungleich sein und dadurch voneinander unterschieden werden können. Dies könnte jedoch zu einem falschen Ergebnis führen, wenn beispielsweise der zweite Vortragende in Wirklichkeit dem*

Institut `DKE` angehört und somit der Wert `DKE` für das Element `institut` vorhanden sein müsste, dieser bei der Überprüfung des Constraints jedoch fehlt. In diesem Fall wären die beiden ersten Vortragenden wertgleich, wodurch der Constraint möglicherweise nicht mehr erfüllt wäre.

Eine weitere Möglichkeit ist, das Problem zu vermeiden, indem beispielsweise zusätzliche strukturelle Einschränkungen mit einem Schema definiert werden oder durch die Syntax der Constraints die entsprechenden Einschränkungen spezifiziert werden. Auch dadurch wird keine zufriedenstellende Lösung ermöglicht, vor allem weil solche strukturellen Anforderungen bei XML Daten nicht vorausgesetzt werden sollten (siehe Vergleichsmerkmal 1). [49]

Es ist somit wichtig, dass ein IC-Ansatz auf dieses Problem eingeht und eine Möglichkeit für eine korrekte Umsetzung bietet [33]. Dies bedeutet, dass die Constraints auch für jene Datenobjekte verwendet werden können, für die manche Datenwerte fehlen. Daraus ergibt sich das nächste Vergleichsmerkmal der Ansätze:

Vergleichsmerkmal 9: *Der Ansatz gewährleistet eine korrekte Umsetzung von Integritätsbedingungen in XML Dokumenten mit nicht vorhandenen Datenobjekten.*

Werden Constraints für XML Dokumente mit nicht vorhandenen Datenobjekten verwendet, können sich die Ergebnisse je nach Sichtweise von *dne* und *unk* unterscheiden. Während beispielsweise ein Constraint unter der Annahme von *dne* erfüllt ist, könnte dies unter der Annahme von *unk* nicht der Fall sein.

Das Vergleichsmerkmal 9 wird daher unterteilt in:

Vergleichsmerkmal 9.1: *Der Ansatz gewährleistet eine korrekte Umsetzung von Integritätsbedingungen in XML Dokumenten mit nicht vorhandenen Datenobjekten unter der Annahme von dne.*

Vergleichsmerkmal 9.2: *Der Ansatz gewährleistet eine korrekte Umsetzung von Integritätsbedingungen in XML Dokumenten mit nicht vorhandenen Datenobjekten unter der Annahme von unk.*

Die Sichtweise von *no info* wird in diesen Vergleich der Ansätze nicht mit einbezogen, da keinerlei Informationen über das Nichtvorhandensein der Datenobjekte zur Verfügung stehen. Dadurch kann nicht festgestellt werden, ob die Datenwerte eigentlich vorhanden

sind, jedoch momentan nicht zur Verfügung stehen, oder ob generell kein Wert für diese Datenobjekte vorhanden ist. [49, 67] Darüber hinaus führt ein Constraint unter der Annahme von *no info* zu demselben Ergebnis wie *unk*. Hierauf wird bei der Angabe der Kontrollbeispiele dieser Vergleichsmerkmale nochmals eingegangen.

In Zusammenhang mit nicht vorhandenen Datenobjekte ist noch eine weitere Abgrenzung von Bedeutung. Diese betrifft die Voraussetzungen, damit ein Constraint erfüllt ist. Es wird unterschieden zwischen *Strong Satisfaction* und *Weak Satisfaction*. Bei einer *Strong Satisfaction* wird das XML Dokument "vervollständigt", d.h. fehlende Werte werden mit einer imaginären Vervollständigung ersetzt. *Strong Satisfaction* wird von einem Constraint erfüllt, wenn alle möglichen Vervollständigungen des XML Dokuments den Constraint erfüllen. *Weak Satisfaction* würde hingegen bedeuten, dass zumindest eine dieser Vervollständigungen den Constraint erfüllen muss. [49]

Unter der Annahme von *dne* und für das Vergleichsmerkmal 9.1 ist diese Unterscheidung nicht relevant, da Werte die in Wirklichkeit nicht vorhanden sind, nicht durch eine Vervollständigung ersetzt werden können [49]. Wird jedoch von *unk* ausgegangen, ist diese Unterscheidung sehr wohl von Bedeutung. [83]

Im Zuge dieser Diplomarbeit und dem Vergleich von IC-Ansätzen wird für das Vergleichsmerkmal 9.2 von *Strong Satisfaction* ausgegangen. Dadurch können Update-Anomalien vermieden werden, sodass Constraints, die vor einem Update erfüllt waren, es auch nach dem Update noch sind. Dies vereinfacht auch die Wartung der Daten. [83] Nachfolgendes Beispiel soll dies verdeutlichen:

Beispiel 4.11: *Wiederum wird der XML Baum in Abbildung 4.24 verwendet, der die Werte dreier Vortragenden zeigt. Ein Element* `vt` *besteht aus den Unterelementen* `name` *und* `institut`*, wobei beim zweiten Vortragenden kein Element* `institut` *vorhanden ist. Wird nun von unk ausgegangen, ist zwar in der Wirklichkeit ein Wert für dieses Element vorhanden, momentan jedoch nicht abrufbar. Würde nun ein Schlüssel* $\sigma =$ `(uni.vortragende.vt, {name, institut})` *oder eine Funktionale Abhängigkeit* $\sigma =$ `(uni.vortragende.vt, {name, institut}` $\rightarrow$ `{@id)}` *für diesen XML Baum definiert werden, wären diese unter Weak Satisfaction erfüllt, da zumindest eine der Vervollständigungen den Constraint erfüllt. Dies könnte jedoch ein falsches Ergebnis liefern, falls bei einem nachträglichen Update der Wert* `DKE` *als Unterelement* `institut` *beim zweiten Vortragenden eingefügt wird. Als Strong Satisfaction definiert wäre der Constraint von Anfang an nicht erfüllt. Ebenso zu diesem Ergebnis würde eine Inklusionsabhängigkeit* $\sigma =$ `(uni.lvas.lva.vt, [name, institut])` $\subseteq$ `(vortragende.vt, [name,institut])` *führen. Für*

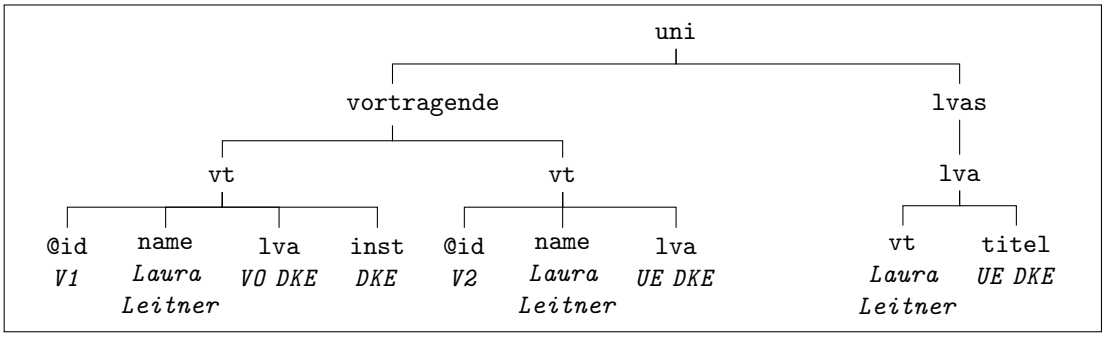

Abbildung 4.25: XML Baum 1 zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte

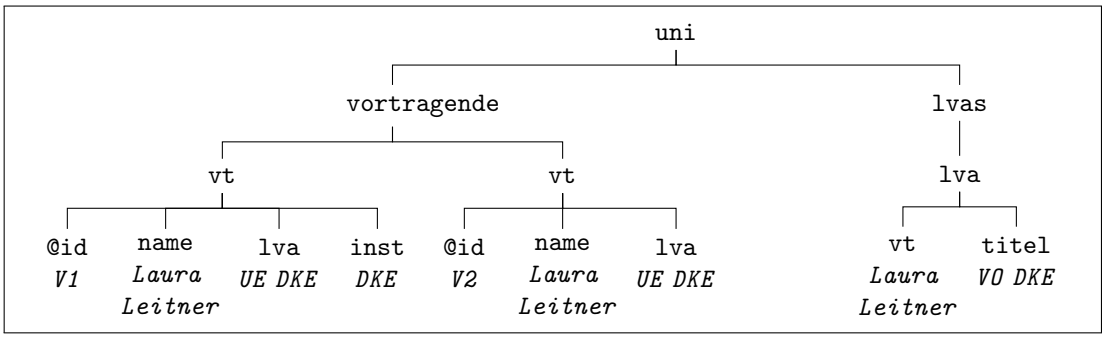

Abbildung 4.26: XML Baum 2 zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte

Strong Satisfaction müssten alle möglichen Vervollständigungen der Werte des zweiten Vortragenden den Constraint erfüllen, während bei Weak Satisfaction nur eine der Vervollständigungen den Constraint erfüllen müsste und dadurch Fehler entstehen könnten.

Um überprüfen zu können, ob ein IC-Ansatz die Vergleichsmerkmale 9.1 und 9.2 erfüllt, werden nachfolgende Kontrollbeispiele anhand der XML Bäume in den Abbildungen 4.25, 4.26 und 4.27 getestet.

Kontrollbeispiel 9.1 - XML Schlüssel:
Ein Vortragender wird identifiziert mit seinem Namen, der von ihm geleiteten Lehrveranstaltung und dem Institut dem er angehört. Der Schlüssel lautet demnach: σ = (uni.vortragende.vt, {name, lva, inst}) *bzw.* σ = (uni.vortragende.vt, {name.S, lva.S, inst.S}) *für jene Ansätze, die auf die Werte der Elementknoten über darunterliegende Textknoten zugreifen. Wird der Schlüssel für den XML Baum in Abbildung 4.25 verwendet, ergeben sich folgende Werte:*

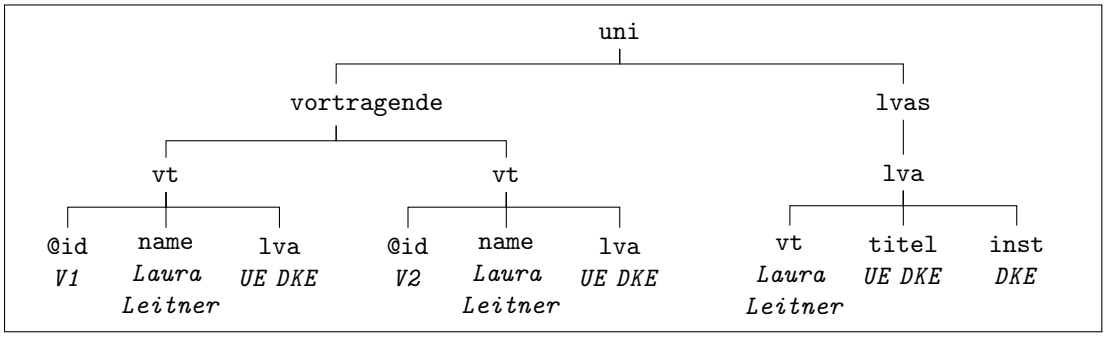

Abbildung 4.27: XML Baum 3 zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte

vt	*name*	*lva*	*inst*
1	*Laura Leitner*	*VO DKE*	*DKE*
2	*Laura Leitner*	*UE DKE*	*–*

1. dne: *Trotz fehlendem* `inst`*-Wert der zweiten Vortragenden ist eine eindeutige Identifizierung der Elemente* `vt` *möglich, da sich bereits die Werte der Elemente* `name` *und* `lva` *unterscheiden. Der Constraint ist somit erfüllt.*

2. unk: *Ebenfalls aus gleichem Grund erfüllt.*

Kontrollbeispiel 9.2 - XML Schlüssel:
Der Schlüssel lautet wiederum σ `= (uni.vortragende.vt, {name, lva, inst})` *bzw.* σ `= (uni.vortragende.vt, {name.S, lva.S, inst.S})`*, wird aber nun verwendet für den XML Baum in Abbildung 4.26. Hier ergeben sich folgende Werte:*

vt	*name*	*lva*	*inst*
1	*Laura Leitner*	*UE DKE*	*DKE*
2	*Laura Leitner*	*UE DKE*	*–*

1. dne: *Erfüllt, da auch in der Realität kein Wert für das Element* `inst` *der zweiten Vortragenden vorhanden ist und sich die Wertemengen {Laura Leitner, UE DKE, DKE} und {Laura Leitner, UE DKE, -} unterscheiden.*

2. unk: *Nicht erfüllt, da die zweite Vortragende in der Realität einem Institut angehört, im Moment jedoch nicht bekannt ist, welchem. Da dieses identisch mit dem Institut der ersten Vortragenden sein könnte, ist eine eindeutige Identifizierung nicht möglich.*

Kontrollbeispiel 9.3 - XML Schlüssel:
Der Schlüssel lautet nochmals σ = (uni.vortragende.vt, {name, lva, inst}) *bzw.* σ = (uni.vortragende.vt, {name.S, lva.S, inst.S}) *und wird anhand des XML Baums in Abbildung 4.27 getestet. Es ergeben sich folgende Werte:*

vt	*name*	*lva*	*inst*
1	*Laura Leitner*	*UE DKE*	*–*
2	*Laura Leitner*	*UE DKE*	*–*

1. dne: *Nicht erfüllt, da sich die vorhandenen Werte der Elemente* `name` *und* `lva` *der beiden Vortragenden nicht voneinander unterscheiden und bei beiden kein Wert für das Element* `inst` *vorhanden ist.*

2. unk: *Nicht erfüllt, da, obwohl in der Realität Werte für das Element* `inst` *beider Vortragenden vorhanden sind, nicht bekannt ist, ob sich diese unterscheiden.*

Bei diesen drei Kontrollbeispielen fällt bereits auf, dass es kein Beispiel gibt, bei dem der Constraint unter der Annahme von *unk* erfüllt ist, unter *dne* jedoch nicht. Der Grund dafür ist, dass ein Schlüssel, der bereits unter *dne* nicht erfüllt ist, auch nicht erfüllt sein kann, wenn Werte zwar in der Realität vorhanden, momentan aber nicht verfügbar sind *(unk)*. Schließlich kann nicht mit Sicherheit ausgeschlossen werden, ob die fehlenden Werte mit vorhandenen Werten identisch sind und die Schlüsselbedingung dadurch verletzt wäre.

Selbiges würde unter der Annahme von *no info* gelten. Hierbei kann nicht ausgeschlossen werden, ob Werte in der Realität vorhanden sind, und, falls dies der Fall ist, ob diese gleich sind wie vorhandene Werte im XML Dokument. Das Kontrollbeispiel 9.1 wäre dennoch erfüllt, weil sich bereits die vorhandenen Werte der Elemente `name` und `lva` unterscheiden. Für das Kontrollbeispiel 9.2 ist dies jedoch nicht der Fall. Die Werte der Elemente `name` und `lva` sind für beide Vortragende gleich, ein `inst`-Wert fehlt jedoch. Um falsche Ergebnisse zu vermeiden, muss davon ausgegangen werden, dass auch die zweite Vortragende dem Institut "DKE" angehören könnte und die Werte beider Vortragenden somit gleich wären. Die Schlüsselbedingung wäre demnach verletzt. Dies entspricht dem Ergebnis von *unk*. Ebenfalls unter *no info* nicht erfüllt ist das Kontrollbeispiel 9.3, da dieses weder unter *dne* noch *unk* erfüllt sein kann. Die Ergebnisse unter *no info* entsprechen somit den Ergebnissen von *unk*.

Kontrollbeispiel 9.4 - XML FDs:
Die ID eines Vortragenden wird bestimmt durch seinen Namen, der von ihm gegebenen

Lehrveranstaltung und seinem Institut. Die Funktionale Abhängigkeit lautet somit: σ = (uni.vortragende.vt, {name, lva, inst} → {@id}) bzw. σ = (uni.vortragende.vt, {name.S, lva.S, inst.S} → {@id}). Wird dieser Constraint für den XML Baum in Abbildung 4.25 verwendet, ergeben sich folgende Werte:

vt	*name*	*lva*	*inst*	*@id*
1	*Laura Leitner*	*VO DKE*	*DKE*	*V1*
2	*Laura Leitner*	*UE DKE*	*–*	*V2*

1. dne: *Erfüllt, da die IDs der Vortragenden trotz fehlendem* `inst`*-Wert eindeutig bestimmt werden können (aufgrund unterschiedlicher Werte für das Element* `lva`*).*

2. unk: *Ebenfalls aus gleichem Grund erfüllt.*

Kontrollbeispiel 9.5 - XML FDs:
Die Funktionale Abhängigkeit σ = (uni.vortragende.vt, {name, lva, inst} → {@id}) bzw. σ = (uni.vortragende.vt, {name.S, lva.S, inst.S} → {@id}) wird nun für den XML Baum in Abbildung 4.26 verwendet. Dies ergibt folgende Kombinationen der Werte für `name`*,* `lva` *und* `inst`*:*

vt	*name*	*lva*	*inst*	*@id*
1	*Laura Leitner*	*UE DKE*	*DKE*	*V1*
2	*Laura Leitner*	*UE DKE*	*–*	*V2*

1. dne: *Erfüllt, da die zweite Vortragende auch in der Realität keinem Institut angehört und sich die beiden Vortragenden dadurch unterscheiden*

2. unk: *Nicht erfüllt, weil die Vortragende zwar in der Realität einem Institut angehört, im Moment jedoch nicht bekannt ist welchem. Würde es sich mit dem Institut der ersten Vortragenden decken, wäre die Funktionale Abhängigkeit verletzt.*

Kontrollbeispiel 9.6 - XML FDs:
Die Funktionale Abhängigkeit lautet nochmals σ = (uni.vortragende.vt, {name, lva, inst} → {@id}) bzw. σ = (uni.vortragende.vt, {name.S, lva.S, inst.S} → {@id}). Beim XML Baum in Abbildung 4.27 führt dies zu folgenden Werten:

vt	*name*	*lva*	*inst*	*@id*
1	*Laura Leitner*	*UE DKE*	*–*	*V1*
2	*Laura Leitner*	*UE DKE*	*–*	*V2*

1. dne: *Nicht erfüllt, da auch in der Wirklichkeit kein Wert für das Element* `inst` *vorhanden ist und sich die vorhandenen Werte der beiden Vortragenden nicht unterscheiden. Dadurch ist eine eindeutige Bestimmung der ID nicht möglich.*

2. unk: *Nicht erfüllt, da die in der Realität existierenden Werte für das* `inst`*-Element nicht vorhanden sind, sich die vorhandenen Werte nicht unterscheiden und somit die ID nicht eindeutig bestimmt werden kann.*

Ebenso wie bei Schlüsselbedingungen kann auch zur Überprüfung von Ansätzen für Funktionale Abhängigkeiten kein Beispiel gefunden werden, dass unter der Annahme von *dne* nicht erfüllt ist, unter *unk* jedoch schon. Des Weiteren würde die *no info* Interpretation wiederum zu demselben Ergebnis führen wie *unk*, sodass auch für Funktionale Abhängigkeiten keine Unterscheidung der Ergebnisse für *unk* und *no info* angegeben wurde.

Kontrollbeispiel 9.7 - XML InclDs:
Die Daten eines Vortragenden einer Lehrveranstaltung (Elemente `vt, titel` *und* `inst`*) sollen auch im Verzeichnis aller Vortragenden (Teilbaum* `vortragende`*) vorhanden sein. Die Inklusionsabhängigkeit lautet:* σ `= (uni.lvas.lva, [vt, titel, inst])` $\subseteq$ `(vortragende.vt, [name, lva, inst])` *bzw.* σ `= (uni.lvas.lva, [vt.S, titel.S, inst.S])` $\subseteq$ `(vortragende.vt, [name.S, lva.S, inst.S])`*. Wird dieser Constraint für den XML Baum in Abbildung 4.25 verwendet, ergeben sich folgende Werte:*

Teilbaum lvas			*Teilbaum vortragende*		
lva.vt	*lva.titel*	*lva.inst*	*vt.name*	*vt.lva*	*vt.inst*
Laura Leitner	*UE DKE*	*–*	*Laura Leitner*	*UE DKE*	*–*
			Laura Leitner	*UE DKE*	*DKE*

1. dne: *Erfüllt, da die* `inst`*-Werte der Vortragenden im Teilbaum* `lvas` *bei der ersten Vortragenden im Teilbaum* `vortragende` *auch nicht vorhanden sind, sich die vorhandenen Werte jedoch gleichen.*

2. unk: *Nicht erfüllt, da nicht bekannt ist, ob der im Teilbaum* `lvas` *momentan nicht vorhandene Werte für das Element* `inst` *in Wirklichkeit mit dem Wert* `DKE` *der zweiten Vortragenden im Teilbaum* `vortragende` *identisch ist*

Kontrollbeispiel 9.8 - XML InclDs:
Wird der Constraint eben vorgestellte Constraint σ `= (uni.lvas.lva, [vt, titel,`

inst]) ⊆ (vortragende.vt, [name, lva, inst]) bzw. *σ = (uni.lvas.lva, [vt.S, titel.S, inst.S]) ⊆ (vortragende.vt, [name.S, lva.S, inst.S])* anhand des XML Baums in Abbildung 4.26 getestet, resultieren folgende Werte:

Teilbaum lvas			*Teilbaum vortragende*		
lva.vt	*lva.titel*	*lva.inst*	*vt.name*	*vt.lva*	*vt.inst*
Laura Leitner	*VO DKE*	–	*Laura Leitner*	*UE DKE*	*DKE*
			Laura Leitner	*UE DKE*	–

1. dne: *Nicht erfüllt, weil bereits die vorhandenen Werte der Vortragenden im Teilbaum* `lvas` *nicht im Teilbaum* `vortragende` *vorhanden sind.*

2. unk: *Nicht erfüllt, da selbst die vorhandenen Werte im Teilbaum* `lvas` *nicht im Teilbaum* `vortragende` *vorhanden sind und dadurch eine Teilmengenbeziehung nicht gegeben ist.*

Bei der Kontrolle von Ansätzen für Inklusionsabhängigkeiten werden nur zwei Beispiele benötigt, weil es entweder möglich ist, dass der Constraint unter *dne* erfüllt ist und unter *unk* nicht, oder er ist sowohl unter *dne* als auch *unk* nicht erfüllt. Die Möglichkeit, dass eine Inklusionsabhängigkeit unter *unk* erfüllt ist, gibt es nicht. Der Grund dafür ist, dass nicht bestimmt werden kann, ob ein im Moment nicht vorhandener Wert auf der LHS einem vorhandenen oder momentan ebenfalls nicht vorhandenen Wert der RHS entsprechen würde und die Inklusionsabhängigkeit somit erfüllt wäre. Zu demselben Ergebnis würde die Sichtweise von *no info* führen, da weder festgestellt werden kann, ob in der Realität ein Wert vorhanden ist, noch ob dieser – falls vorhanden – gleich einem vorhandenen Wert ist. Dies entspricht wiederum dem Ergebnis von *unk*.

Mehrfach vorhandene Datenobjekte

Im Gegensatz zu nicht vorhandenen Datenobjekten ist es auch möglich, dass Datenobjekte mit demselben Label innerhalb eines Elementknotens mehrfach vorhanden sind. In diesem Zusammenhang werden oft die Begriffe Einmaligkeit bzw. Einzigartigkeit (engl.: uniqueness) und Identifizierung genannt, welche vor allem für Schlüsselbeziehungen von Bedeutung sind. Identifizierung bedeutet, dass eine genau definierte Menge von Knoten einen anderen Knoten identifiziert. Um die Einmaligkeit zu gewährleisten, ist es nicht erlaubt, dass Knoten, die zum Identifizieren verwendet werden, mehrfach im Element vorkommen. Wird beispielsweise eine Lehrveranstaltung (Element `lva`) mit seinen Unterelementen `vt` und `inst` identifiziert, wäre es nicht erlaubt, dass zwei Elemente `vt` mit

demselben Wert im Element `lva` vorhanden sind. Im Gegensatz zu XML Dokumenten ist diese Unterscheidung bei relationalen Datenbanken nicht notwendig, da mehrfach vorhandene Tupel nicht erlaubt sind. [49]

Für semistrukturierte und hierarchisch aufgebaute Daten ist dies eigentlich nicht vorgesehen. Vielmehr schränkt es die flexible und irreguläre Datenstruktur von XML Daten ein und sollte deshalb vermieden werden (siehe Abschnitt 2.2.2). Daher ist es wichtig, dass ein IC-Ansatz dies unterstützt und eine Möglichkeit bietet, Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten korrekt umzusetzen. [7] Das nächste Vergleichsmerkmal lautet somit:

Vergleichsmerkmal 10: *Der Ansatz gewährleistet eine korrekte Umsetzung von Integritätsbedingungen in XML Dokumenten mit mehrfach vorhandenen Datenobjekten.*

Eine korrekte Umsetzung bedeutet hierbei, dass vom IC-Ansatz eine Möglichkeit geschaffen wird, die der Realität entsprechenden, richtigen Kombinationen der Datenwerte herauszufinden. Dadurch soll garantiert werden, dass nur jene Kombinationen für die Constraints verwendet werden, die der Originalsemantik und somit der Wirklichkeit entsprechen. [49] Das folgende Beispiel soll dies verdeutlichen:

Beispiel 4.12: *Der XML Baum in Abbildung 4.28 zeigt, dass für eine Lehrveranstaltung zwei Vortragende vorhanden sind, weil die LVA von zwei Vortragenden abwechselnd abgehalten wird. Werden nun für einen Constraint die Elemente `titel, vn, nn` als Fields verwendet, muss garantiert werden, dass nur die richtigen Kombinationen von Vorname und Nachname der Vortragenden zum Wertvergleich verwendet werden. Richtig sind die Wertekombinationen {VO DKE, Laura, Leitner} und {VO DKE, Felix, Schnell}, falsch wären hingegen {VO DKE, Felix, Leitner} und {VO DKE, Laura, Schnell}.*

Um herauszufinden, ob ein IC-Ansatz das Vergleichsmerkmal erfüllt, werden nachfolgende Kontrollbeispiele verwendet.

Kontrollbeispiel 10.1 - XML Schlüssel:
Eine Lehrveranstaltung wird durch ihren Titel, dem Namen des Vortragenden und dem Semester identifiziert. Der Schlüssel lautet: σ = `(uni.lvas.lva, {titel, vt.name, vt.sem})` bzw. σ = `(uni.lvas.lva, {titel.S, vt.name.S, vt.sem..S})` für Ansätze, die auf die Elementwerte über darunterliegende Textknoten zugreifen. Dies führt bei der Anwendung am XML Baum in Abbildung 4.29 zu folgenden Werten:

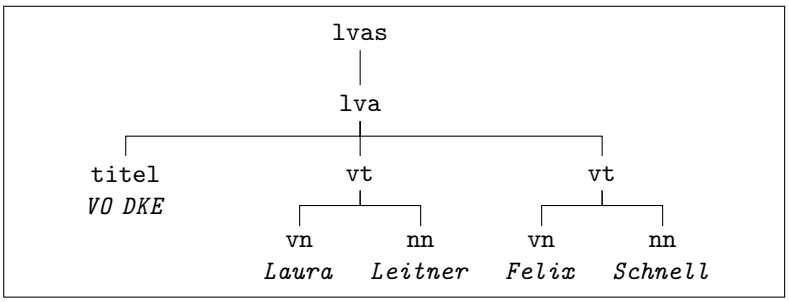

Abbildung 4.28: XML Baum zur Darstellung von richtigen Wertekombinationen bei mehrfach vorhandenen Datenobjekten

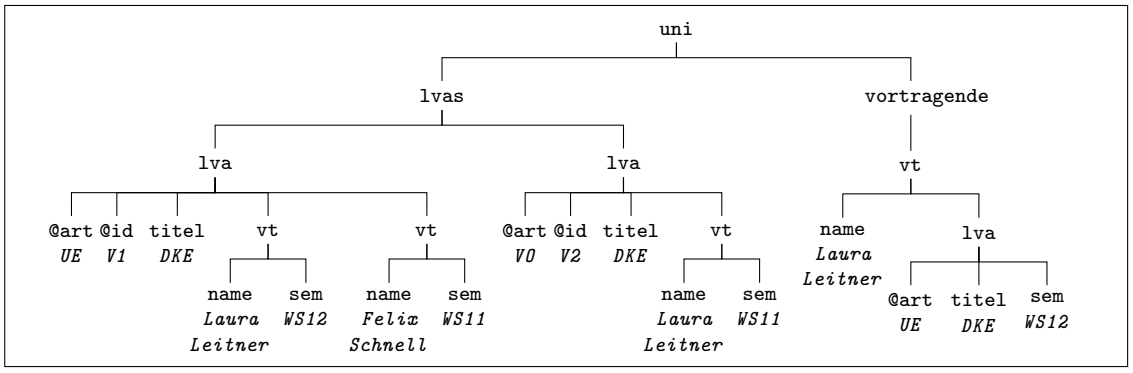

Abbildung 4.29: XML Baum 1 zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten

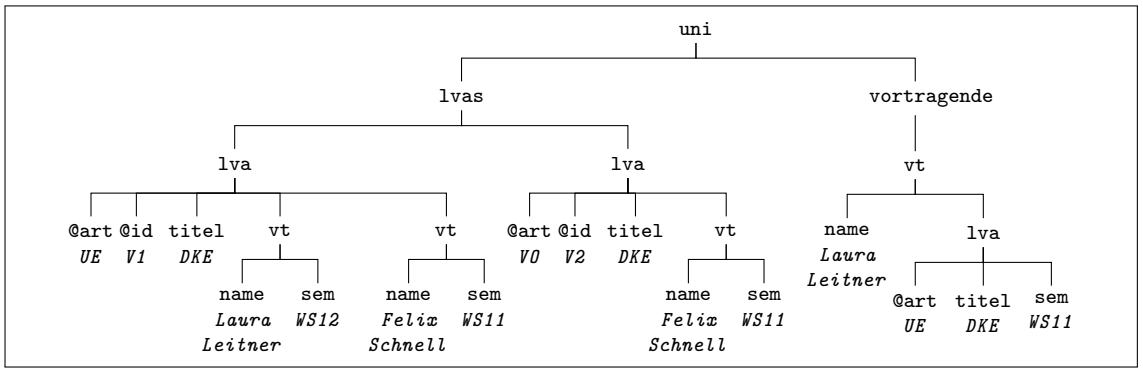

Abbildung 4.30: XML Baum 2 zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten

lva	*titel*	*vt.name*	*vt.sem*
1	*DKE*	*Laura Leitner*	*WS12*
	DKE	*Felix Schnell*	*WS11*
2	*DKE*	*Laura Leitner*	*WS11*

Bei semantisch korrekter Verknüpfung der Werte der durch die Pfade `titel, vt.name` *und* `vt.sem` *erreichbaren Knoten kann eine Lehrveranstaltung eindeutig identifiziert werden. Der Constraint ist somit für diesen XML Baum erfüllt. Würden die Werte hingegen nicht korrekt verknüpft werden, könnte dies zu folgendem Ergebnis führen:*

lva	*titel*	*vt.name*	*vt.sem*
1	*DKE*	*Laura Leitner*	*WS11*
	DKE	*Felix Schnell*	*WS12*
2	*DKE*	*Laura Leitner*	*WS11*

Da die Wertekombination {DKE, Laura Leitner, WS11} zweimal vorkommt, wäre eine eindeutige Identifizierung der Lehrveranstaltungen nicht mehr möglich und der Constraint dadurch nicht erfüllt.

Kontrollbeispiel 10.2 - XML Schlüssel:
Der Schlüssel `σ = (uni.lvas.lva, {titel, vt.name, vt.sem})` *bzw.* `σ = (uni.lvas.lva, {titel..S, vt.name.S, vt.sem.S})` *wird nun für den XML Baum in Abbildung 4.30 verwendet. Hierbei ergeben sich folgende Werte:*

lva	*titel*	*vt.name*	*vt.sem*
1	*DKE*	*Laura Leitner*	*WS12*
	DKE	*Felix Schnell*	*WS11*
2	*DKE*	*Felix Schnell*	*WS11*

Wie in dieser Aufstellung ersichtlich ist, sind die für diesen Constraint relevanten Werte in der zweiten und dritten Zeile gleich, obwohl es sich um verschiedene `lva`*-Elemente handelt. Die Schlüsselbedingung ist somit nicht erfüllt. Würden jedoch falsche Wertekombinationen verwendet werden, könnte dies zu folgendem Ergebnis führen:*

lva	*titel*	*vt.name*	*vt.sem*
1	*DKE*	*Laura Leitner*	*WS11*
	DKE	*Felix Schnell*	*WS12*
2	*DKE*	*Felix Schnell*	*WS11*

Hier wäre der Constraint erfüllt, was jedoch nicht die Wirklichkeit widerspiegeln und zu einem falschen Ergebnis führen würde.

Kontrollbeispiel 10.3 - XML FDs:
Die ID einer Lehrveranstaltung wird bestimmt durch ihren Titel, dem Namen des Vortragenden und dem Semester in dem sie abgehalten wird. Dementsprechend lautet die Funktionale Abhängigkeit σ = `(uni.lvas.lva, {titel, vt.name, vt.sem}` $\rightarrow$ `{@id})` *bzw.* σ = `(uni.lvas.lva, {titel.S, vt.name.S, vt.sem.S}` $\rightarrow$ `{@id})`*. Für den XML Baum in Abbildung 4.29 ergeben sich dadurch folgende Werte:*

titel	*vt.name*	*vt.sem*	*@id*
DKE	*Laura Leitner*	*WS12*	*V1*
DKE	*Felix Schnell*	*WS11*	*V1*
DKE	*Laura Leitner*	*WS11*	*V2*

Da bei semantisch korrekter Kombination der Datenwerte die ID einer Lehrveranstaltung eindeutig bestimmt werden kann, ist der Constraint erfüllt. Eine falsche Verknüpfung der Werte könnte zu folgendem Ergebnis führen:

titel	*vt.name*	*vt.sem*	*@id*
DKE	*Laura Leitner*	*WS11*	*V1*
DKE	*Felix Schnell*	*WS12*	*V1*
DKE	*Laura Leitner*	*WS11*	*V2*

Mit diesem Ergebnis wäre der Constraint nicht erfüllt, da die Wertemenge {DKE, Felix Schnell, WS11} zweimal vorkommt, sie jedoch verschiedene IDs bestimmen sollten.

Kontrollbeispiel 10.4 - XML FDs:
Die Funktionale Abhängigkeit σ = `(uni.lvas.lva, {titel, vt.name, vt.sem}` $\rightarrow$ `{@id})` *bzw.* σ = `(uni.lvas.lva, {titel.S, vt.name.S, vt.sem.S}` $\rightarrow$ `{@id})` *wird nun für den XML Baum in Abbildung 4.30 verwendet. Dies führt zu folgenden Werten:*

titel	*vt.name*	*vt.sem*	*@id*
DKE	*Laura Leitner*	*WS12*	*V1*
DKE	*Felix Schnell*	*WS11*	*V1*
DKE	*Felix Schnell*	*WS11*	*V2*

Diese Aufstellung zeigt, dass die ID der Lehrveranstaltungen mit diesen Werten nicht eindeutig bestimmt werden kann. Die Funktionale Abhängigkeit ist somit für diesen XML Baum nicht erfüllt. Bei einer falschen Kombination der Werte könnte folgende Auflistung herauskommen:

titel	*vt.name*	*vt.sem*	*@id*
DKE	*Laura Leitner*	*WS11*	*V1*
DKE	*Felix Schnell*	*WS12*	*V1*
DKE	*Felix Schnell*	*WS11*	*V2*

Der Constraint wäre hier erfüllt, da sich die Werte {DKE, Laura Leitner, WS11}, {DKE, Felix Schnell, WS12} und {DKE, Felix Schnell, WS11} unterscheiden würden und die IDs somit eindeutig bestimmt werden könnten. Dies würde jedoch zu einem falschen Ergebnis führen.

Kontrollbeispiel 10.5 - XML InclDs:
Der Name und die Daten der Lehrveranstaltung eines Vortragenden in der Liste aller Vortragenden muss auch in der Liste aller Lehrveranstaltungen vorhanden sein. Die Inklusionsabhängigkeit lautet somit: σ `= (uni.vortragende.vt [name, lva.titel, lva.sem])` $\subseteq$ `(uni.lvas.lva, [vt.name, titel, vt.sem])` *bzw.* σ `= (uni.vortragende.vt [name.S, lva.titel.S, lva.sem.S])` $\subseteq$ `(uni.lvas.lva, [vt.name.S, titel.S, vt.sem.S])`*. Wenn dieser Constraint für den XML Baum in Abbildung 4.29 verwendet wird, ergeben sich nachfolgende Werte:*

Teilbaum `vortragende` ***Teilbaum*** `lvas`

name	*lva.titel*	*lva.sem*	*vt.name*	*titel*	*vt.sem*
Laura Leitner	*DKE*	*WS12*	*Laura Leitner*	*DKE*	*WS12*
–	–	–	*Felix Schnell*	*DKE*	*WS11*
–	–	–	*Laura Leitner*	*DKE*	*WS11*

Die Inklusionsabhängigkeit ist bei semantisch korrekter Verknüpfung der Datenwerte erfüllt. Würden die Werte jedoch falsch kombiniert werden, könnte dies zu folgendem Ergebnis führen:

Teilbaum `vortragende`			***Teilbaum*** `lvas`		
name	*lva.titel*	*lva.sem*	*vt.name*	*titel*	*vt.sem*
Laura Leitner	*DKE*	*WS12*	*Laura Leitner*	*DKE*	*WS11*
–	–	–	*Felix Schnell*	*DKE*	*WS12*
–	–	–	*Laura Leitner*	*DKE*	*WS11*

Hier wurden die Werte der Elemente `name` *und* `sem` *im Teilbaum* `lvas` *falsch verknüpft, sodass die Wertekombination {Laura Leitner, DKE, WS12} dort nicht mehr vorhanden ist und die Inklusionsabhängigkeit nicht erfüllt ist. Dieses Ergebnis würde jedoch nicht die Wirklichkeit widerspiegeln.*

Kontrollbeispiel 10.6 - XML InclDs:
Die Inklusionsabhängigkeit σ `= (uni.vortragende.vt [name, lva.titel, lva.sem])` $\subseteq$ `(uni.lvas.lva, [vt.name, titel, vt.sem])` *bzw.* σ `= (uni.vortragende.vt [name.S, lva.titel.S, lva.sem.S])` $\subseteq$ `(uni.lvas.lva, [vt.name.S, titel.S, vt.sem.S])` *wird nun für den XML Baum in Abbildung 4.30 verwendet, was zu folgenden Werten führt:*

Teilbaum `vortragende`			***Teilbaum*** `lvas`		
name	*lva.titel*	*lva.sem*	*vt.name*	*titel*	*vt.sem*
Laura Leitner	*DKE*	*WS11*	*Laura Leitner*	*DKE*	*WS12*
–	–	–	*Felix Schnell*	*DKE*	*WS11*
–	–	–	*Felix Schnell*	*DKE*	*WS11*

Da die Wertekombination {Laura Leitner, UE DKE, WS11} im Teilbaum `vortragende` *vorhanden ist, im Teilbaum* `lvas` *jedoch fehlt, ist bei semantisch korrekter Umsetzung die Inklusionsabhängigkeit nicht erfüllt. Würden die Werte jedoch falsch kombiniert werden, könnte dies zu folgendem Ergebnis führen:*

Teilbaum `vortragende`			***Teilbaum*** `lvas`		
name	*lva.titel*	*lva.sem*	*vt.name*	*titel*	*vt.sem*
Laura Leitner	*DKE*	*WS11*	*Laura Leitner*	*DKE*	*WS11*
–	–	–	*Felix Schnell*	*DKE*	*WS12*
–	–	–	*Felix Schnell*	*DKE*	*WS11*

Hier wäre der Constraint erfüllt, weil die Wertemenge {Laura Leitner, UE DKE, WS11} sowohl im Teilbaum `vortragende` *als auch im Teilbaum* `lvas` *vorhanden ist. Dies würde jedoch zu einem falschen Ergebnis führen.*

4.2 Zusammenfassung

In diesem Kapitel wurden elf Merkmale zum Vergleich von Ansätzen für XML Integritätsbedingungen vorgestellt. Darüber hinaus gäbe es natürlich noch weitere, speziellere Bereiche, anhand derer IC-Ansätze verglichen werden könnten. Es wäre beispielsweise möglich, noch genauer auf die Funktionen einzelner Pfadsprachen einzugehen und deren erweiterte Ausdruckskraft im Zusammenhang mit Constraints zu analysieren. Es könnte auch die Definition der Knoten- und Wertgleichheit erweitert werden und z.B. verschiedene Datentypen in den Wertvergleich integriert werden. Ein weiteres mögliches Vergleichsmerkmal wäre die Erhaltung der Originaldaten, wenn beispielsweise relationale Daten in XML Daten umgewandelt werden, sowie eine Analyse des Reasonings. [33]

Reasoning kann übersetzt werden als logische Argumentation oder Beweisführung. Die wichtigsten Fragen betreffen dabei das Konsistenz- und Implikationsproblem. Das Konsistenzproblem beschäftigt sich mit der Frage, ob für eine angegebene Menge von XML Constraints zumindest ein XML Dokument existiert, dass diese Constraints erfüllt. Ist auch eine Schemaspezifikation vorhanden, wird auch diese berücksichtigt. Das XML Dokument soll somit den Anforderungen der Schemaspezifikation entsprechen und die Constraints erfüllen. Das Implikationsproblem bezieht sich auf die Frage, ob eine angegebene Menge von Constraint einen weiteren Constraint impliziert, d.h., ob ein XML Dokument, das eine Menge von Constraints erfüllt, auch einen weiteren Constraint erfüllt. Die logische Implikation von Constraints kann mit Hilfe von Algorithmen oder einer fehlerfreien und vollständigen Menge von Inferenzregeln bewiesen werden.[33]

Diese Liste könnte natürlich beliebig erweitert werden. Dies würde jedoch den Rahmen dieser Diplomarbeit sprengen. Des Weiteren sollte eine Möglichkeit geschaffen werden, Vergleichsmerkmale zu finden, die in möglichst vielen IC-Ansätzen bearbeitet bzw. umgesetzt wurden. Aus diesem Grund wurden die Vergleichsmerkmale auf die in diesem Kapitel vorgestellten Bereiche beschränkt.

Die Tabelle in Abbildung 4.31 zeigt nochmals alle angegebenen und im weiteren Verlauf verwendete Vergleichsmerkmale im Überblick.

4.3 Related Work

Im Folgenden werden nun einige Arbeiten anderer Autoren vorgestellt, in denen verschiedenen Ansätze verglichen werden. Hierbei wird deutlich, dass die zum Vergleich verwen-

Vergleichsmerkmale	
1	**Unabhängigkeit von einer Schemasprache.** Der Ansatz ist nicht an eine Schemasprache gebunden.
2	**Selektion der Knoten** Der Ansatz unterstützt die Verwendung eines Kleene Operators sowie Pfade, die mit Elementknoten enden.
3	**Baummodell - Dokumentordnung** Der Ansatz verwendet ein Baummodell, das das Einhalten der Dokumentordnung gewährleistet.
4	**Baummodell - Gemischter Inhalt** Der Ansatz verwendet ein Baummodell, das gemischten Inhalt unterstützt.
5	**Arität** Der Ansatz ermöglicht die Definition von n-ären Integritätsbedingungen.
6	**Wertvergleich von gemischtem Inhalt** Der Ansatz ermöglicht die Definition von Integritätsbedingungen mit Wertvergleich von gemischtem Inhalt.
7	**Baumbasierter Wertvergleich** Der Ansatz ermöglicht die Definition von Integritätsbedingungen mit baumbasiertem Wertvergleich.
8	**Geltungsbereich** Der Ansatz ermöglicht die Definition von relativen Integritätsbedingungen.
9.1	**Nicht vorhandene Datenobjekte unter der Annahme von *dne*** Der Ansatz gewährleistet eine korrekte Umsetzung von Integritätsbedingungen in XML Dokumenten mit nicht vorhandenen Datenobjekten unter der Annahme von *dne*.
9.2	**Nicht vorhandene Datenobjekte unter der Annahme von *unk*** Der Ansatz gewährleistet eine korrekte Umsetzung von Integritätsbedingungen in XML Dokumenten mit nicht vorhandenen Datenobjekten unter der Annahme von *unk*.
10	**Mehrfach vorhandene Datenobjekte** Der Ansatz gewährleistet eine korrekte Umsetzung von Integritätsbedingungen in XML Dokumenten mit mehrfach vorhandenen Datenobjekten.

Abbildung 4.31: Vergleichsmerkmale

deten Merkmale in den Ansätzen meist auf einige wenige beschränkt sind.

Ahmat et al. [4] gehen in ihrem Ansatz auf die Wichtigkeit einer klaren Definition von Wertgleichheit ein, vor allem auch im Vergleich zu relational Datenbanken. Darüber hinaus wird die verwendete Pfadsprache diskutiert, wobei auch die Verwendung von Kleene Operatoren aufgrund der hierarchischen Datenstruktur von XML als sinnvoll erachtet wird. Ebenfalls von Bedeutung ist laut Ahmat et al. die Möglichkeit zur Definition von

relativen Constraints.

Arenas und Libkin [7] vergleichen Ansätze für Funktionale Abhängigkeiten und Schlüsselbedingungen hinsichtlich Dokumentordnung, Bindung an eine Schemasprache und der Eingrenzung des Geltungsbereichs. Erweitert wurde dies von Arenas, Fan und Libkin [5] um die Arität und die verwendete Pfadsprache.

Barcelò, Libkin und Reutter [9] konzentrieren sich auf die Behandlung von XML Dokumenten mit nicht vorhandenen Datenobjekten und gehen auch auf die Bedeutung der Pfadsprache bei der Definition von Constraints ein.

Buneman et al. [19, 20] betonen die Wichtigkeit der Form des Wertvergleichs (baumbasierter Wertvergleich statt Einschränkung auf einfachen String-Vergleich), des Geltungsbereichs, der Unabhängigkeit von einer Schemasprache, der Wahl einer passenden Pfadsprache, der Arität und des Umganges mit nicht vorhandenen Datenobjekten.

Chen, Liao und Gao [23, 24] gehen auf IC-Ansätze in XML Dokumenten mit nicht vorhandenen Datenobjekten ein sowie auf Null-Werte, die Unabhängigkeit von einer Schemaspezifikation, den Geltungsbereich und die verwendete Form des Wertvergleichs.

Fan [33] zeigt einen Überblick von einigen bis 2005 entwickelten Ansätzen von XML Constraints. Im Speziellen geht er auf die Bedeutung von relativen Constraints ein. Wichtige Punkte in dieser Analyse sind auch die Auswirkung von komplexen Pfadsprachen bei der Definition von Constraints, die Arität und mögliche Wechselwirkungen zwischen Constraints und DTDs.

Fan und Libkin [34] beschäftigen sich ebenfalls mit der Zusammenwirkung von DTDs und XML Integritätsbedingungen und wie sie sich gegenseitig (negativ) beeinflussen können.

Fan und Siméon [35] zeigen eine Analyse von ersten Ansätzen von XML Integritätsbedingungen (vor 2003), auch im Vergleich zu Integritätsbedingungen in relationalen und objektorientierten Datenbanksystemen. Des Weiteren beschäftigen sie sich mit der Verwendung von Schemasprachen als Grundlage eines Ansatzes und der Auswahl einer geeigneten Pfadsprache bzw. der verwendeten Pfadfunktionen bei der Definition der Constraints.

Ferrarotti et al. [36] zeigen Beispiele, für die die Verwendung eines (Single-labeled) Wildcards bei der Definition der Constraints sinnvoll ist bzw. dass bestimmte Constraints ohne Wildcard nicht abgebildet werden können.

Hartmann et al. [38] vergleichen verschiedene Ansätze für XML Schlüssel, im Speziellen gehen sie dabei auf die Bereiche absolute und relative Schlüssel, *Weak* und *Strong Satisfaction*, die verwendete Pfadsprache, die Verwendung von Wildcards, das Einhalten der Dokumentordnung sowie die Definition der Wertgleichheit ein.

Hartmann und Link [40] beschäftigten sich mit der Verwendung von Listen bei der Definition von Mehrwertigen Abhängigkeiten in traditionellen Datenbanken, verweisen aber auch auf dessen Wichtigkeit für XML Constraints. Auch von Hartmann, Link und Schewe [44] wurde dieses Thema behandelt. Sie beschreiben einige Anwendungsbereiche bei denen die Verwendung von Listen sinnvoll ist.

Hartmann und Link [42] vergleichen einige Ansätze u.a. bezüglich der Bindung an eine Schemasprache, der Einschränkung des Geltungsbereichs und der verwendeten Definition der (Wert-) Gleichheit.

Hartmann und Trinh [45] vergleichen verschiedene IC-Ansätze auf Grundlage der verwendeten Definition der Wertgleichheit, der Arität und dem Umgang mit Constraints in XML Dokumenten mit teilweise nicht vorhandenen Datenobjekten. Hierbei wird auch auf die Interpretationen *unk* und *dne* eingegangen. Im Besonderen werden hierbei die Ansätze von Arenas und Libkin [7] und Vincent et al. [79, 83, 84] verglichen. Ebenso wurden diese Ansätze in [82] verglichen.

Karlinger [49] gibt einen sehr guten Überblick über bisher vorgestellte IC-Ansätze. Dazu wurden die Ansätze zuerst in Kategorien eingeteilt (siehe auch Abschnitt 3.3.1). Verglichen wurden Ansätze dabei vorwiegend aufgrund ihrer Vorgehensweise bei XML Dokumenten mit mehrfach oder nicht vorhandenen Datenobjekten und ob eine Möglichkeit geboten wird, die ursprüngliche relationale Datensemantik bei der Umwandlung in XML zu erhalten. Im Speziellen behandelt Karlinger auch die Unterscheidung von *Strong* und *Weak Satisfaction* und den damit verbundenen Eigenschaften Einmaligkeit und Identifizierung sowie den Sichtweisen *unk, dne* und *no info* beim Umgang mit nicht vorhandenen Datenobjekten. Eingegangen wird auch auf den baumbasierten Wertvergleich und die verschiedenen Möglichkeiten, diesen umzusetzen. Diese Bereiche wurden auch von Karlinger, Vincent und Schrefl [50, 51] diskutiert, wobei auch auf die Wichtigkeit der Wahl einer geeigneten Pfadsprache und der Definition des Geltungsbereichs eines Constraints eingegangen wird.

Liu, Vincent und Liu [59] gehen im Speziellen auf die Definition von relativen XML Constraints ein und zeigen Beispiele, welche mit Hilfe von absoluten Constraints nicht

abgebildet werden könnten.

Lv und Yan [60, 61] vergleichen verschiedene Ansätze von Funktionalen Abhängigkeiten in XML, wobei hauptsächlich auf den Vergleich von Tree-Tuple-basierten, Pfad-basierten und Sub-Graph-basierten Ansätzen eingegangen wird. In [60] wurde auch auf den Umgang mit nicht vorhandenen Datenobjekten und Nullwerten, den Geltungsbereich von Constraints, die verwendete Pfadsprache (u.a. dass der Pfad mit einem Elementknoten enden kann und die Verwendung des Wildcard-Symbols), die Bindung an eine Schemasprache, die verwendete Definition der Wertgleichheit und die Arität eingegangen.

Mok [64] beschäftigt sich mit dem Umgang von gemischtem Inhalt bei der Verwendung von Constraints und der Abhängigkeit eines Ansatzes von einer Schemasprache.

Shahriar und Liu [74, 75] gehen auf die Definition des Geltungsbereichs bei Constraints ein. Sie zeigen des Weiteren, dass eine semantisch korrekte Abbildung, v.a. im Zusammenhang mit mehrfach vorhandenen Datenobjekten, für die Umsetzung der Constraints von großer Wichtigkeit ist. Analysiert wird auch die gewählte Form des Wertvergleichs.

Vincent, Liu und Liu [79, 83] vergleichen Ansätze in Bezug auf *Strong* und *Weak Satisfaction*, den Umgang mit nicht vorhandenen Datenobjekten in XML Dokumenten, die verwendete Definition der Wertgleichheit zweier Knoten, die Pfadsprache (ein Pfad kann auch mit einem Elementknoten enden) und die Bindung des Ansatzes an eine Schemasprache.

Vincent, Liu und Mohania [85] vergleichen vordergründig die beiden Ansätze von Arenas und Libkin [7] und Vincent et al. [79, 83, 84] u.a. auf Basis des Umgangs mit nicht vorhandenen Datenobjekten, der Bindung an eine DTD, der Dokumentordnung, gemischtem Inhalt und der Arität.

Vincent, Schrefl und Liu [86] beschäftigen sich mit der Definition von Constraints die mit einem Elementknoten enden und beschreiben darüber hinaus die Vorteile von n-ären Constraints und der Ungebundenheit an eine Schemaspezifikation.

Wang und Topor [88] schränken ihre Definition von Funktionalen Abhängigkeit auf XML Dokumente ein, bei denen die Elementknoten nicht geordnet sein müssen und gemischter Inhalt nicht erlaubt ist (u.a. um Redundanzen zu vermeiden). Sie erwähnen jedoch, dass diese Eigenschaften auch für XML relevant sind und für zukünftige Arbeiten in diesem Bereich von großer Bedeutung sind. Des Weiteren vergleichen sie verschiedene Ansätze auf Grundlage der verwendeten Definition des Wertvergleichs und Nullwerten, die Bin-

dung an eine Schemaspezifikation und die Eingrenzung von Constraints auf bestimmte Teilbäume (absolute vs. relative Constraints).

Die Tabelle in Abbildung 4.32 fasst das Ergebnis nochmals zusammen.

All diesen Studien liegt zu Grunde, dass sie nur bestimmte Aspekte im Zusammenhang mit der Definition von XML Constraints analysieren und nur einzelne Formen von Integritätsbedingungen bearbeiten. So gehen beispielsweise Lv und Yan [61] in ihrer Analyse nur auf Funktionale Abhängigkeiten ein, Schlüssel und Inklusionsabhängigkeiten werden jedoch nicht behandelt. Hartmann et al. [38] beschränken sich hingegen auf Schlüsselbeziehungen und vernachlässigen Funktionale Abhängigkeiten und Inklusionsabhängigkeiten.

Um eine Möglichkeit zu schaffen, die von den meisten Autoren als wichtig erachteten Eigenschaften gemeinsam zur Analyse von IC-Ansätzen verwenden zu können, wurden die soeben vorgestellten Vergleichsmerkmale ausgearbeitet.

4.4 Verglichene Ansätze

In den folgenden Kapiteln werden verschiedene IC-Ansätze anhand der Vergleichsmerkmale analysiert. Dafür wurden einige bekannte Ansätze ausgewählt, welche sich teilweise grundlegend voneinander unterscheiden.

Im Kapitel 5 wird auf den vom World Wide Web Consortium im Zuge der Spezifikation von XML Schema [76] veröffentlichten Ansatz eingegangen. Dieser bedient sich der *Selector/Field*-Methode. Schlüssel und Fremdschlüssel werden dabei mit den in XML Schema definierten Typen `key` und `keyref` spezifiziert. Zur Definition des *Selectors* werden XPath-Ausdrücke [12] verwendet.

Als nächstes wird im Kapitel 6 ein Ansatz von Buneman et al. [18, 19, 20] vorgestellt. Es handelt sich dabei um einen der ersten Ansätze zur Definition von XML Schlüsseln. Dabei wird eine von Buneman et al. vorgestellte Pfadsprache zur Definition des *Context Path* (schränkt den Geltungsbereich ein), des *Target Set* (dem zu identifizierenden Element) und der *Key Paths* (die identifizierenden Datenobjekte) verwendet.

Der nächste Ansatz ist von Arenas und Libkin [6, 7] und wird im Kapitel 7 vorgestellt. Es handelt sich dabei um einen sogenannten *Tree Tuple* Ansatz zur Definition von Funktionalen Abhängigkeiten. XML Dokumente werden dabei als ausgeflachte relationale Tupel abgebildet. Dadurch wird gewährleistet, dass nur semantisch korrekte Wertekombinationen

Quelle	Schemasprache	Pfadsprache	Dokumentordnung	Gemischter Inhalt	Arität	Wertvergleich	Geltungsbereich	Nicht vorhandene Datenobjekte	Mehrfach vorhandene Datenobjekte
Ahmad et al. [4]		✓				✓	✓		
Arenas und Libkin [7]	✓		✓				✓		
Arenas, Fan und Libkin [5]	✓	✓	✓		✓		✓		
Barcelò, Libkin und Reutter [9]		✓						✓	
Buneman et al. [19, 20]	✓	✓			✓	✓	✓	✓	
Chen, Liao und Gao [23, 24]	✓					✓	✓	✓	
Fan [33]	✓	✓			✓		✓		
Fan und Libkin [34]	✓								
Fan und Siméon [35]	✓	✓							
Ferrarotti et al. [36]		✓							
Hartmann et al. [38]		✓	✓			✓	✓	✓	
Hartmann und Link [40, 44]			✓						
Hartmann und Link [42]	✓					✓	✓		
Hartmann und Trinh [45]					✓	✓		✓	
Karlinger [49]						✓		✓	✓
Karlinger, Vincent und Schrefl [50, 51]		✓				✓	✓	✓	✓
Liu, Vincent, Liu [59]							✓		
Lv und Yan [60, 61]	✓	✓			✓	✓	✓	✓	
Mok [64]	✓			✓					
Shrariar und Liu [74, 75]						✓	✓		✓
Vincent, Liu und Liu [79, 83]	✓	✓				✓		✓	
Vincent, Liu und Mohania [85]	✓		✓	✓	✓			✓	
Vincent, Schrefl und Liu [86]	✓	✓			✓				
Wand und Topor [88]	✓		✓	✓		✓	✓	✓	

Abbildung 4.32: Related Work

in einem Tupel vorhanden sind und Constraints auch im Fall von mehrfach vorhandenen Datenobjekten korrekt umgesetzt werden können.

Im Kapitel 8 wird ein Ansatz von Fan und Libkin [34] vorgestellt. Er befasst sich mit der Definition von n-ären Schlüsseln, Fremdschlüsseln und Inklusionsabhängigkeiten. Voraussetzung dafür ist das Vorhandensein einer DTD, da zur Definition die Elementtypen bzw. deren Attribute verwendet werden.

Im Kapitel 9 wird ein von Lee, Ling und Low [58] veröffentlichter Ansatz zur Definition von Funktionalen Abhängigkeiten vorgestellt. Voraussetzung dafür ist, dass der *Header* (zur Definition des Geltungsbereichs), die LHS und die RHS der Funktionalen Abhängigkeit einer sogenannten *lineage* entsprechen müssen, welche verglichen werden kann mit einer Abstammungslinie.

Das Kapitel 10 befasst sich mit einem Ansatz von Vincent, Liu und Liu [83, 84] bzw. Vincent, Liu und Mohania [85]. Die Autoren definieren dazu die sogenannte *Closest*-Eigenschaft, um semantisch korrekte Wertekombinationen zu ermöglichen. Des Weiteren unterscheiden sie zwischen *Weak* und *Strong* Constraints, wobei sie in ihrer Definition von *Strong Constraints* ausgehen.

Der nächste Ansatz ist von Vincent et al. [86] und wird im Kapitel 11 vorgestellt. In [49] wird dieser Ansatz als *Intersection Path Approach* bezeichnet. Dabei muss der Pfad zum *Selector* die Schnittmenge der Pfade zu den *Fields* bilden.

Als nächstes wird im Kapitel 12 ein Ansatz von Hartmann und Link [39] bzw. Hartmann, Link und Kirchberg [43] analysiert. Hierbei werden Funktionale Abhängigkeiten mit Hilfe von Subgraphen definiert, d.h., die LHS und RHS müssen Subgraphen des *Selector*-Knotens sein.

Im Kapitel 13 wird anschließend ein Ansatz von Mok [64] vorgestellt. Funktionale Abhängigkeiten werden dabei mit Hilfe von *Templates*, d.h. einer Hypothese und einer Konklusion, definiert.

Abschließend wird im Kapitel 14 ein Ansatz von Karlinger [49] vorgestellt. Es ist ein sogenannter *Enhanced Closest Node*-Ansatz zur Definition von Schlüsseln und Fremdschlüsseln, wodurch in XML Dokumenten mit mehrfach vorhandenen Datenobjekten eine semantisch korrekte Wertekombination garantiert wird. Des Weiteren wird mit der Definition der *Maximum Combination of Field Nodes* eine Möglichkeit geboten, Constraints für XML Dokumente mit fehlenden Datenobjekten zu definieren.

Bei diesen Ansätzen handelt es sich um eine kleine Auswahl, da eine Analyse sämtlicher in der Literatur vorhandener Ansätze den Rahmen dieser Arbeit sprengen würde. Vielmehr soll veranschaulicht werden, wie sehr sich die Ansätze teilweise voneinander unterscheiden und auf welche Art die Vergleichsmerkmale zur Analyse von Ansätzen verwendet werden können.

Des Weiteren wird darauf hingewiesen, dass, sofern keine andere Quelle angegeben wird, jene Quellen zur Analyse eines Ansatzes verwendet wurden, die in der Einleitung des jeweiligen Ansatzes angeführt sind.

Teil II

Vergleich von Ansätzen wertbasierter Integritätsbedingungen in XML

Kapitel 5

Ansatz des World Wide Web Consortium

Das World Wide Web Consortium (W3C) veröffentlichte eine Ansatz zur Definition von Schlüsseln und Fremdschlüsseln mit Hilfe der Schemasprache XML Schema [15, 32, 76]. Dieser wird im folgenden Kapitel erläutert.

5.1 Kurzbeschreibung

Der Ansatz vom W3C verwendet zur Definition von Schlüsseln und Fremdschlüsseln die Typen `key` bzw. `keyref` sowie die Typen `selector` und `field` eines XML Schemas. Dazu wird folgende Syntax angegeben:

```
<key
id = ID
name = NCName
{weitere Attribute, die nicht im Schema-Namensraum sind . . .}>
Inhalt: (annotation?, (selector, field+))
</key>

<keyref
id = ID
name = NCName
refer = QName
```

```
{weitere Attribute, die nicht im Schema-Namensraum sind . . .}>
Inhalt: (annotation?, (selector, field+))
</keyref>

<selector
id = ID
xpath = Untermenge der XPath-Mächtigkeit
{weitere Attribute, die nicht im Schema-Namensraum sind . . .}>
Inhalt: (annotation?)
</selector>

<field
id = ID
xpath = Untermenge der XPath-Mächtigkeit
{weitere Attribute, die nicht im Schema-Namensraum sind . . .}>
Inhalt: (annotation?)
</field>
```

Die Angabe eines `id`-Wertes ist hier optional, wohingegen die Unterelemente `selector` und `field` vorhanden sein müssen. Ein `selector`-Element muss dabei genau einmal vorhanden sein. Die Anzahl der `field`-Elemente ist lediglich darin beschränkt, dass zumindest ein Element pro Schlüssel bzw. Fremdschlüssel vorhanden sein muss, es können aber auch unbeschränkt viele sein (Häufigkeit "+"). Des Weiteren muss der Name des Schlüssels oder Fremdschlüssels in Form eines Attributes angegeben werden sowie bei Fremdschlüsseln auch der Verweis auf einen Schlüssel mit Hilfe des Attributes `refer`.

5.2 Überprüfung der Vergleichsmerkmale

Im Folgenden wird der Ansatz anhand der Vergleichsmerkmale in Kapitel 4 analysiert.

5.2.1 Unabhängigkeit von einer Schemasprache

Das W3C geht in ihrem Ansatz vom Vorhandensein der Schemasprache XML Schema aus. Das Vergleichsmerkmal 1 ist demnach nicht erfüllt.

5.2.2 Pfadsprache

Zur Definition des *Selctors* und der *Fields* werden XPath-Ausdrücke [12] verwendet. Ein *Selector*-Pfad muss dabei mit einem Elementknoten enden, wohingegen ein *Field*-Pfad mit einem Element- oder Attributknoten enden kann. Des Weiteren können Kleene Operatoren verwendet werden.

Wird diese Vorgehensweise nun anhand der Kontrollbeispiele 2.1 und 2.2 geprüft, wird der Schlüssel folgendermaßen definiert:

```
<xs:key name="vt_key">
  <xs:selector xpath="//vt"/>
  <xs:field xpath="name"/>
</xs:key>
```

Es wird definiert, dass ein Vortragender mit dessen Namen identifiziert werden kann, egal wo im XML Baum er sich befindet. Im XML Baum in Abbildung 4.2 ist dieser Schlüssel erfüllt, wohingegen er im XML Baum in Abbildung 4.3 nicht erfüllt ist, da mehrere Vortragende mit dem Namen *"Laura Leitner"* vorhanden sind.

Dies entspricht für beide Kontrollbeispiele dem gewünschten Ergebnis und das Vergleichsmerkmal 2 ist daher für Schlüsselbedingungen erfüllt.

Als nächstes wird dieser Ansatz für Fremdschlüsseln anhand der Kontrollbeispiele 2.5 und 2.6 analysiert. Der Fremdschlüssel wird demnach definiert als:

```
<xs:key name="vt_key">
  <xs:selector xpath="//vortragende/vt"/>
  <xs:field xpath="name"/>
</xs:key>

<xs:keyref name="vt_foreignkey" refer="vt_key">
  <xs:selector xpath="//lva/vt"/>
  <xs:field xpath="name"/>
</xs:keyref>
```

Für den XML Baum in Abbildung 4.2 ist dieser Constraint nicht erfüllt, weil der `name`-Wert der Vortragenden `Laura Leitner` im Teilbaum `vortragende` nicht vorhanden ist. Dies entspricht dem gewünschten Ergebnis. Ebenfalls korrekt umgesetzt wird das Kon-

trollbeispiel 2.6. Im XML Baum in Abbildung 4.3 ist die Fremdschlüssel-Beziehung erfüllt, weil der `name`-Wert *"Laura Leitner"* auch im Teilbaum `vortragende` vorhanden ist.

Das Vergleichsmerkmal 2 ist demnach auch für Fremdschlüssel erfüllt.

5.2.3 Baummodell

XML Schema ist auf den Spezifikationen von XML Information Set [26] und XPath [12] aufgebaut. Das XML Information Set [26] definiert dabei die Kinder eines Elementknotens folgendermaßen:

Definition 5.1: *"An ordered list of child information items, in document order. This list contains element, processing instruction, unexpanded entity reference, character, and comment information items, one for each element, processing instruction, reference to an unprocessed external entity, data character, and comment appearing immediately within the current element. If the element is empty, this list has no members."*

Aus dieser Definition geht hervor, dass die Dokumentordnung im XML Dokument eingehalten wird. Der XML Baum in Abbildung 4.6 kann daher korrekt abgebildet werden und das Vergleichsmerkmal 3 ist erfüllt.

Des Weiteren wird von XML Schema das Vorhandensein von gemischtem Inhalt unterstützt, da in der XML Spezifikation [17] folgendes definiert wurde:

Definition 5.2: *An element type has mixed content when elements of that type may contain character data, optionally interspersed with child elements.*

Der XML Baum in Abbildung 4.8 kann somit ebenfalls korrekt abgebildet werden und das Vergleichsmerkmal 4 ist demnach erfüllt.

5.2.4 Arität

Wie bereits oben definiert wurde, kann ein `key`- bzw. `keyref`-Element zumindest ein aber auch beliebig viele `field`-Elemente beinhalten. Dadurch wird eine Definition von n-ären Constraints ermöglicht.

Für die Kontrollbeispiele 5.1 und 5.2 wird nachfolgender Schlüssel definiert:

```
<xs:key name="vt_key">
  <xs:selector xpath="/uni/vortragende/vt"/>
  <xs:field xpath="@vn"/>
  <xs:field xpath="@nn"/>
</xs:key>
```

Durch die Angabe von zwei `field`-Elementen kann der n-äre Schlüssel definiert werden. Für das Kontrollbeispiel 5.1 und den XML Baum in Abbildung 4.9 wird der Constraint korrekt umgesetzt, da die Wertemengen *{Laura, Leitner}* und *{Felix, Schnell}* verschieden sind und die Vortragenden somit eindeutig identifiziert werden können.

Im Gegensatz dazu ist die Schlüsselbedingung für den XML Baum in Abbildung 4.10 nicht erfüllt. Da dies ebenfalls dem gewünschten Ergebnis entspricht, ist das Vergleichsmerkmal 5 für Schlüsselbedingungen erfüllt.

Als nächstes wird die Anwendung dieses Ansatzes bei Fremdschlüsseln analysiert. Dazu wird folgendes Schema definiert:

```
<xs:key name="vt_key">
  <xs:selector xpath="/uni/vortragende/vt"/>
  <xs:field xpath="@vn"/>
  <xs:field xpath="@nn"/>
</xs:key>

<xs:keyref name="vt_foreignkey" refer="vt_key">
  <xs:selector xpath="/uni/lvas/lva/vt"/>
  <xs:field xpath="@vn"/>
  <xs:field xpath="@nn"/>
</xs:keyref>
```

Auch hierbei wird eine Umsetzung ermöglicht, da mit Hilfe mehrerer `field`-Elemente n-äre Fremdschlüssel definiert werden können. Die Kontrollbeispiele 5.5 und 5.6 können demnach korrekt umgesetzt werden und das Vergleichsmerkmal 5 ist auch für Fremdschlüssel erfüllt.

5.2.5 Wertvergleich

XML Schema verwendet eine Form des Wertvergleichs, die von einfachem String-Vergleich zu unterscheiden ist. So sind die Werte "3.0" und "3" gleich, wenn sie beide als Zahlen definiert sind, jedoch nicht gleich, wenn sie beide als String-Werte oder ein Wert als String-Wert und einer als Zahl definiert sind.

Beim Vergleich von Elementknoten wird ein baumbasierter Wertvergleich verwendet. Dadurch wird ein Wertvergleich von gemischtem Inhalt nicht ermöglicht. Das Vergleichsmerkmal 6 ist demnach nicht erfüllt.

Um zu testen, ob der in XML Schema definierte baumbasierte Wertvergleich zum richtigen Ergebnis führt, werden die Kontrollbeispiele 7.1 und 7.2 bzw. 7.5 und 7.6 verwendet. Dazu werden folgende Schlüssel und Fremdschlüssel definiert:

```
<xs:key name="vt_key">
  <xs:selector xpath="/uni/vortragende/vt"/>
  <xs:field xpath="personendaten"/>
</xs:key>

<xs:keyref name="vt_foreignkey" refer="vt_key">
  <xs:selector xpath="/uni/lvas/lva/vt"/>
  <xs:field xpath="personendaten"/>
</xs:keyref>
```

Für den XML Baum in Abbildung 4.17 ist der Schlüssel erfüllt, da für beide Elemente `personendaten` der gesamte Unterbaum zum Vergleich herangezogen wird. Diese Unterbäume unterscheiden sich sowohl in den Baumstrukturen als auch in den Werten. Das Kontrollbeispiel 7.1 wird somit korrekt umgesetzt. Im XML Baum in Abbildung 4.19 ist der Schlüssel jedoch nicht erfüllt, weil die Unterbäume beider Vortragenden gleich sind und sie somit nicht eindeutig identifiziert werden können. Da beide Kontrollbeispiele korrekt umgesetzt wurden, ist das Vergleichsmerkmal 7 für Schlüsselbedingungen erfüllt.

Der oben angeführte Fremdschlüssel wird nun für die XML Bäume in den Abbildungen 4.17 und 4.18 verwendet. Hierbei ist der Fremdschlüssel erfüllt, da der Unterbaum des `personendaten`-Elements im Teilbaum `lvas` auch im Teilbaum `vortragende` vorhanden ist. Nicht erfüllt ist der Fremdschlüssel jedoch für die XML Bäume in den Abbildungen 4.19 und 4.20. Hier sind die Daten der Vortragenden des Teilbaums `lvas` nicht im

Teilbaum `vortragende` vorhanden.

Da wiederum beide Kontrollbeispiele korrekt umgesetzt wurden, ist das Vergleichsmerkmal 7 für Fremdschlüssel ebenfalls erfüllt.

5.2.6 Geltungsbereich von Constraints

Der Geltungsbereich eines Constraints kann in XML Schema eingegrenzt werden, indem er innerhalb der Elementtyp-Definition definiert wird. Für die Kontrollbeispiele 8.1 und 8.2 sowie 8.5 und 8.6 müssen die Schlüssel- und Fremdschlüsseldefinitionen somit in die Definition des Elementtyps `institut` inkludiert werden. Dies sieht folgendermaßen aus[11]:

```
<xs:element name="uni">
  <xs:element name="institut">
    <xs:element name="name"/>
    <xs:element name="lvas">
      ...
    </xs:element>
    <xs:element name="vortragende">
      <xs:element name="vt">
        <xs:element name="name"/>
        <xs:element name="lva">
          <xs:attribute name="id"/>
          <xs:attribute name="titel"/>
        </xs:element>
        <xs:key name="vt_key">
          <xs:selector xpath="lva"/>
          <xs:field xpath="@titel"/>
        </xs:key>
      </xs:element>
    </xs:element>

    <xs:key name="lva_key">
      <xs:selector xpath="lvas/lva"/>
```

[11] Das XML Schema wird hier verkürzt dargestellt und zeigt lediglich die für dieses Kontrollbeispiel relevanten Bereiche.

```
        <xs:field xpath="@titel"/>
      </xs:key>

      <xs:keyref name="lva_foreignkey" refer="lva_key">
        <xs:selector xpath="vortragende/vt/lva"/>
        <xs:field xpath="@titel"/>
      </xs:keyref>

    </xs:element>
  </xs:element>
```

Die Schlüsselbedingung σ = (uni.institut.vortragende.vt, (lva, {@titel})) wird im Schema bei der Typdefinition des Elements vt eingefügt, sodass dieser Schlüssel nur innerhalb eines Vortragenden gültig ist. Für den XML Baum in Abbildung 4.22 ist der Schlüssel erfüllt, weil die @titel-Attribute innerhalb eines vt-Elements eindeutig sind. Für den XML Baum in Abbildung 4.23 ist die Schlüsselbedingung jedoch nicht erfüllt, da im ersten Teilbaum vt zwei Lehrveranstaltungen mit demselben Titel vorhanden sind und somit nicht eindeutig identifiziert werden können. Dies entspricht für beide Kontrollbeispiele dem gewünschten Ergebnis und das Vergleichsmerkmal 8 ist für Schlüssel erfüllt.

Der Fremdschlüssel bzw. die Inklusionsabhängigkeit σ = (uni.institut, (vortragende.vt.lva, [@titel]) $\subseteq$ (lvas.lva, [@titel])) wird ebenfalls im Schema inkludiert, jedoch bei der Typdefinition des Elements institut. Der Grund dafür ist, dass innerhalb eines Instituts der Titel einer Lehrveranstaltung im Verzeichnis der Vortragenden ebenfalls im Verzeichnis der Lehrveranstaltungen vorhanden sein muss. Dies wird anhand der Kontrollbeispiele 8.5 und 8.6 getestet. Für den XML Baum in Abbildung 4.22 ist der Constraint wie gewünscht erfüllt. Die @titel-Werte der Lehrveranstaltungen im Teilbaum vortragende eines Instituts sind auch im Teilbaum lvas vorhanden. Für den XML Baum in Abbildung 4.23 ist der Fremdschlüssel jedoch nicht erfüllt. Innerhalb eines Instituts sind die @titel-Werte der Lehrveranstaltungen im Teilbaum vortragende nicht im Teilbaum lvas vorhanden. Dies entspricht dem gewünschten Ergebnis.

Da auch hier beide Kontrollbeispiele korrekt umgesetzt wurden, ist das Vergleichsmerkmal 8 für Fremdschlüssel ebenfalls erfüllt.

5.2.7 Nicht vorhandene Datenobjekte

Um die im XML Schema definierten Schlüsseln und Fremdschlüsseln zu erfüllen, muss jeder *Field*-Knoten für jeden *Selector*-Knoten genau einmal vorhanden sein. Es ist somit nicht erlaubt, dass Datenobjekte nicht vorhanden sind.

Wird dies für Schlüsselbedingungen nun anhand der Kontrollbeispiele 9.1, 9.2 und 9.3 getestet, führt dies zu folgenden Ergebnissen:

Das Kontrollbeispiel 9.1 wird weder unter der Annahme von *dne* noch *unk* korrekt umgesetzt, da die Schlüsselbedingung wegen fehlendem `inst`-Wert der zweiten Vortragenden nicht erfüllt ist.

Ebenfalls nicht erfüllt ist der Schlüssel im XML Baum in Abbildung 4.26. Auch hier fehlt der `inst`-Wert der zweiten Vortragenden. Dieses Ergebnis ist für *dne* nicht gefordert, für *unk* jedoch schon. Das Kontrollbeispiel 9.2 wird somit unter der Annahme von *unk* korrekt umgesetzt.

Für den XML Baum in Abbildung 4.27 ist die Schlüsselbedingung ebenfalls nicht erfüllt, da die `inst`-Werte beider Vortragenden fehlen. Dieses Ergebnis ist unter der Annahme von *dne* und *unk* gefordert, sodass das Kontrollbeispiel 9.3 korrekt umgesetzt wird.

Zum Testen der Fremdschlüsseln werden nun die Kontrollbeispiele 9.7 und 9.8 verwendet. Für den XML Baum in Abbildung 4.25 ist der Fremdschlüssel nicht erfüllt, da sowohl im Teilbaum `lvas` als auch im Teilbaum `vortragende` für den Vergleich relevante Werte fehlen. Dies entspricht unter der Annahme von *unk* dem richtigen Ergebnis, für *dne* jedoch nicht.

Für den XML Baum in Abbildung 4.26 soll der Fremdschlüssel gemäß Kontrollbeispiel 9.8 weder unter der Annahme von *dne* noch *unk* erfüllt sein. Da dieses Ergebnis mit der Fremschlüssel-Definition des W3C erreicht wird, wird das Kontrollbeispiel korrekt umgesetzt.

Da weder für *dne* noch *unk* alle Vergleichsmerkmale korrekt umgesetzt werden können, sind die Vergleichsmerkmale 9.1 und 9.2 nicht erfüllt.

5.2.8 Mehrfach vorhandene Datenobjekte

Wie soeben erwähnt, muss jeder *Field*-Knoten für jeden *Selector*-Knoten genau einmal vorhanden sein. Sind *Field*-Knoten mehrfach vorhanden, ist der Constraint nicht erfüllt. Diese Vorgehensweise wird nun anhand der Kontrollbeispiele 10.1 und 10.2 bzw. 10.5 und 10.6 getestet.

Im Kontrollbeispiel 10.1 soll der Schlüssel σ = (`uni.lvas.lva, {titel, vt.name, vt.sem}`) erfüllt sein. Dies wird vom Ansatz des W3C nicht ermöglicht, da auf Grund mehrfach vorhandener Datenobjekte die Schlüsselbedingungen für den XML Baum in Abbildung 4.29 nicht erfüllt ist.

Aus dem gleichen Grund ist die Schlüsselbedingung auch im XML Baum in Abbildung 4.30 nicht erfüllt. Hier ist dies jedoch gewünscht und das Kontrollbeispiel 10.5 wird korrekt umgesetzt.

Zur Kontrolle der Fremdschlüsseln werden nun die Kontrollbeispiele 10.5 und 10.6 verwendet. Für den XML Baum in Abbildung 4.29 ist der Fremdschlüssel wegen mehrfach vorhandener `titel`- und `sem`-Werte nicht erfüllt. Dies entspricht für das Kontrollbeispiel 10.5 nicht dem geforderten Ergebnis.

Ebenfalls aus gleichen Grund nicht erfüllt ist der Fremdschlüssel für den XML Baum in Abbildung 4.30. Hier ist dies jedoch gefordert und das Kontrollbeispiel 10.6 wird korrekt umgesetzt.

Da nicht alle Kontrollbeispiele richtig umgesetzt werden können, ist das Vergleichsmerkmal 10 nicht erfüllt.

5.3 Zusammenfassung

In Abbildung 5.1 fasst die Ergebnisse der Analyse dieses Ansatzes nochmals zusammen.

Das erste Vergleichsmerkmal wird nicht erfüllt, da der Ansatz das Vorhandensein eines XML Schemas voraussetzt. Bei der Definition der Constraints werden XPath-Ausdrücke [12] verwendet, sodass die Verwendung eines Kleene Operators ermöglicht wird. Es ist auch erlaubt, dass *Field*-Pfade mit einem Element enden, sodass das 2. Vergleichsmerkmal erfüllt ist. XML Schema baut auf der Spezifikationen von XML Information Set [26] und XPath [12] auf. Hierbei wird definiert, dass die Kinder eines Elements geordnet sein müssen. Aus diesem Grund wird das 3. Vergleichsmerkmal erfüllt. Des Weiteren wird

Vergleichsmerkmale		
1	Schema	nicht erfüllt
2	Pfadsprache	erfüllt
3	Baummodel: Dokumentordnung	erfüllt
4	Baummodell: Gemischter Inhalt	erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	erfüllt
8	Geltungsbereich	erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	nicht erfüllt

Abbildung 5.1: Vergleichsmerkmale für den Ansatz vom World Wide Web Consortium

in der XML Spezifikation [17] definiert, dass auch gemischter Inhalt erlaubt ist. Das 4. Vergleichsmerkmal ist demnach ebenfalls erfüllt.

Da bei der Definition von Schlüsseln und Fremdschlüsseln mehrere *Fields* angegeben werden können, ist das 5. Vergleichsmerkmal ebenfalls erfüllt. Des Weiteren werden Elemente auf Grund ihrer Unterbäume verglichen, sodass das 7. Vergleichsmerkmal erfüllt ist, das 6. Vergleichsmerkmal hingegen nicht. Das 8. Vergleichsmerkmal ist ebenfalls erfüllt, weil durch das Einfügen der Schlüsseln und Fremdschlüsseln in die Elementtyp-Definition eine Einschränkung des Geltungsbereichs möglich ist.

Nicht erfüllt sind hingegen die Vergleichsmerkmale 9.1 und 9.2 sowie 10, da jeder *Field*-Knoten für jeden *Selector*-Knoten genau einmal vorhanden sein muss.

Kapitel 6

Ansatz von Buneman et al.

Einer der ersten Ansätze für XML Integritätsbedingungen wurde von Buneman et al. [18, 19, 20] entwickelt. Dieser wird im Folgenden kurz beschrieben und anhand der im Kapitel 4 angegebenen Vergleichsmerkmale analysiert.

6.1 Kurzbeschreibung

Der Ansatz von Buneman et al. beschreibt eine Möglichkeit, XML Schlüssel zu spezifizieren. Diese werden definiert als $\sigma = (Q, \{P_1, \ldots, P_n\})$. Die Abkürzung Q steht für einen Pfadausdruck der von Buneman et al. definierten Pfadsprache (siehe unten) und identifiziert eine Menge von Knoten, bezeichnet als *Target Set*, für welche der Schlüssel gelten soll. $\{P_1, \ldots, P_n\}$ bezeichnet eine Menge von einfachen Pfadausdrücken, auch genannt *Key Paths*, die zu den identifizierenden Knoten führen.

6.2 Überprüfung der Vergleichsmerkmale

Im Folgenden wird erläutert, welche Merkmale von diesem Ansatz erfüllt werden.

6.2.1 Unabhängigkeit von einer Schemasprache

Der Ansatz von Buneman et al. ist nicht an eine Schemasprache gebunden. Schlüssel können daher unabhängig von einer Schemasprache definiert werden. Das erste Vergleichsmerkmal, also die Unabhängigkeit von einer Schemasprache, ist somit erfüllt.

6.2.2 Pfadsprache

Buneman et al. verwenden in ihrem Ansatz die beiden selbst entwickelte Pfadsprachen PL und PL_S. Diese sind folgendermaßen definiert:

Pfadsprache	Syntax
PL_S	p ::= $\epsilon \mid l.p$
PL	q ::= $\epsilon \mid l \mid q.q \mid \_*$

Hierbei bezeichnet ϵ einen leeren Pfad, d.h. eine leere Folge von Labels. *l* bezeichnet ein Element-, Attribut- oder Textlabel (S) und *l.p* bezeichnet die Verkettung von einem Pfad *p* mit einem weiteren Label *l*. Dadurch ermöglicht die Pfadsprache die Definition einfacher Abwärtspfade sowie leere Pfade. Die erweiterte Pfadsprache PL ermöglicht überdies auch die Verkettung von Pfaden ($q.q$) sowie den Kleene Operator $\_*$.

Für die Verwendung der Pfadsprachen zur Definition der Constraints wurden in [18, 20] einige Einschränkungen formuliert. Während für die Pfade in *Q* die Sprache PL verwendet werden kann, ist für die *Key Paths* nur die Verwendung der Pfadsprache PL_S erlaubt. Dies ermöglicht zwar die Definition von Constraints die mit Elementknoten enden, nicht jedoch die Verwendung eine Kleene Operators. Diese Regelung wurde in [19] geändert, sodass auch in den *Key Paths* die Pfadsprache PL verwendet werden kann. Es ist somit auch ermöglicht, Pfade bei der Definition der Constraints zu verwenden, die Kleene Operatoren beinhalten.

Somit kann der in Kontrollbeispiel 2.1 verwendete Schlüssel σ `= (_*.vt, {name})` definiert werden. Des Weiteren ergeben sich bei der Verwendung des Schlüssels für den XML Baum in Abbildung 4.2 dieselben Werte wie im Kontrollbeispiel 2.1 gefordert, sodass dieses korrekt umgesetzt wird.

Gleiches gilt für das Kontrollbeispiel 2.2, da die Vortragenden im XML Baum in Abbildung 4.3 nicht eindeutig identifiziert werden können und dieses Ergebnis für das Kontrollbeispiel vorgesehen ist.

Da beide Kontrollbeispiele korrekt umgesetzt werden können, ist das Vergleichsmerkmal 2 erfüllt.

6.2.3 Baummodell

Der Ansatz basiert auf folgendem Baummodell:

Definition 6.1: *"An XML tree is defined to be T = (V, lab, ele, att, val r), where*

- *V is a set of nodes;*
- *lab is a mapping $V \rightarrow \boldsymbol{E} \cup \boldsymbol{A} \cup \{S\}$ which assigns a label to each node in V; a node v in V is called an element (E node) if lab(v) $\in \boldsymbol{E}$, an attribute (A node) if lab(v) $\in \boldsymbol{A}$, and a text node (S node) if lab(v) = S;*
- *ele and att are partial mappings that define the edge relation of T: for any node v in V,*
  - *if v is an element then ele(v) is a list of elements and text nodes in V and att(v) is a set of attributes in V; for each v' in ele(v) or att(v), v' is called a child of v and we say that there is a (directed) edge from v to v';*
  - *if v ist an attribute or a text node then ele(v) and att(v) are undefined;*
- *val is a partial mapping that assigns a string to each attribute and text node: for any node v in V, if v is an A or S node then val(v) is a string, and val(v) is undefined otherwise;*
- *r is the unique and distinguished root node. An XML tree has a tree structure, i.e., for each v $\in$ V, there is a unique path of edges from root r to v. An XML tree is said to be finite if V is finite."*

Hierbei steht *V* für eine endliche Menge von Knoten und *lab(v)* weißt jedem Knoten *v* ein Label zu. Wenn *v* ein Elementknoten ist, liefert *ele(v)* die Liste aller Unterelemente und Textknoten und *att(v)* die Menge der Attribute von *v*. *val(v)* liefert für jeden Attribut- und Textknoten dessen Stringwert und ist für Elementknoten undefiniert. *r* bezeichnet den Wurzelknoten des Baums.

Aufgrund dieser Definition wird die Dokumentordnung eingehalten, da die Funktion $ele(v)$ eine geordnete Liste aller Element- und Textknoten in v zurückgibt. Der XML Baum in Abbildung 4.6 kann daher korrekt abgebildet werden und das Vergleichsmerkmal 3 wird von diesem Ansatz erfüllt.

Des Weiteren ist das in Buneman et al. verwendete Baummodell angelehnt an das DOM [56], welches erlaubt, dass ein Elementknoten sowohl Unterelemente als auch Textknoten beinhalten kann. Gemischter Inhalt ist somit möglich und der XML Baum in Abbildung 4.8 kann, wie im Kontrollbeispiel 4.1 gefordert, abgebildet werden. Das Vergleichsmerkmal 4 ist deshalb ebenfalls erfüllt

6.2.4 Arität

Der Ansatz in Buneman et al. bedient sich der *Selector / Field* Methode, wobei als Fields eine Menge $\{P_1, \ldots, P_n\}$ von Pfaden verwendet wird. Folglich wird die Definition von n-ären Constraints ermöglicht. Dies wird nun anhand der Kontrollbeispiele 5.1 und 5.2 getestet. Hierfür soll der Schlüssel σ `=(uni.vortragende.vt, {@vn, @nn})` definiert werden.

Wird der Schlüssel für den XML Baum in Abbildung 4.9 verwendet, führt das zu dem Ergebnis, dass die beiden Vortragenden eindeutig anhand der Wertmengen {*Laura, Leitner*} und {*Felix, Schnell*} identifiziert werden können. Dies ist im Kontrollbeispiel 5.1 gefordert, sodass es vom Ansatz korrekt umgesetzt wird.

Für das Kontrollbeispiel 5.2 soll der Schlüssel nun für den XML Baum in Abbildung 4.10 verwendet werden. Hier können die beiden Vortragenden, wie im Kontrollbeispiel vorgesehen, nicht eindeutig identifiziert werden.

Da beide Kontrollbeispiele korrekt umgesetzt werden können, ist das Vergleichsmerkmal 5 erfüllt.

6.2.5 Wertvergleich

Buneman et al. unterstützen zwar die Verwendung von gemischtem Inhalt in ihrem Baummodell (siehe oben), gehen jedoch nicht explizit darauf ein, ob ein Wertvergleich basierend auf der Verkettung von Textwerten für die Definition der Constraints verwendet werden kann. Das Vergleichsmerkmal 6 ist daher nicht erfüllt.

Im Gegensatz dazu wird von Buneman et al. auf den baumbasierten Wertvergleich eingegangen. Dabei werden zwei Knoten x und y als wertgleich bezeichnet, wenn sie in folgenden Punkten übereinstimmen (informell):

- der Menge S an relativen Pfaden zu den Unterknoten,
- der partiellen Funktion von S zu den Knotennamen und
- der partiellen Funktion von S zu Stringwerten.

Zwei Knoten x und y sind somit gleich, wenn die Unterknoten dieselben Pfade relativ zu x bzw. y haben und die Namen und Werte der Unterknoten übereinstimmen.

Um herauszufinden, ob dies zu einem korrekten Ergebnis führt, werden die Kontrollbei-

spiele 7.1 und 7.2 und die Schlüsselbedingung σ = (uni.vortragende.vt, {personendaten}) verwendet. Für den XML Baum in Abbildung 4.17 ergeben sich für die beiden zu vergleichenden personendaten-Elemente folgende Pfade, Knotennamen und Werte:

Vortragende 1			Vortragende 2		
Pfad	Knotenname	Wert	Pfad	Knotenname	Wert
@gebdat	gebdat	01.01.80	@gebdat	gebdat	01.01.55
@status	status	aktiv	@status	status	inaktiv
name	name	–	name	name	–
name.@titel	titel	Dr.	name.vn	vn	Laura
name.vn	vn	Laura	name.nn	nn	Leitner
name.nn	nn	Leitner			

Diese Tabelle zeigt, dass die beiden Vortragenden mit dem in diesem Ansatz verwendeten Wertvergleich eindeutig identifiziert werden können, da sich sowohl die Unterbäume (also Pfade und Knotennamen) als auch die Werte unterscheiden. Da dies im Kontrollbeispiel 7.1 gefordert ist, wird es korrekt umgesetzt.

Für das Kontrollbeispiel 7.2 soll der baumbasierte Wertvergleich für den XML Baum in Abbildung 4.19 verwendet werden. Dies führt zu folgendem Ergebnis:

Vortragende 1			Vortragende 2		
Pfad	Knotenname	Wert	Pfad	Knotenname	Wert
@gebdat	gebdat	01.01.55	@gebdat	gebdat	01.01.55
@status	status	inaktiv	@status	status	inaktiv
name	name	–	name	name	–
name.vn	vn	Laura	name.vn	vn	Laura
name.nn	nn	Leitner	name.nn	nn	Leitner

Die Auflistung zeigt, dass sowohl die relativen Pfade zu den Unterknoten der Elemente personendaten als auch die Knotennamen und Werte beider Vortragenden gleich sind. Eine eindeutige Identifizierung ist somit nicht möglich und die Schlüsselbedingung ist nicht erfüllt. Da dieses Ergebnis für das Kontrollbeispiel 7.2 vorgesehen ist, ist das Vergleichsmerkmal 7 erfüllt.

6.2.6 Geltungsbereich von Constraints

Buneman et al. stellen neben der oben vorgestellten Definition für Schlüsselbedingungen auch eine Definition für relative Schlüssel vor. Dabei wird ein relativer Schlüssel definiert

als σ = *(Q, (Q', {*P_1 *, . . . ,* P_n*}))*. Mit dem Pfad Q wird jener Bereich definiert, in dem der Schlüssel *(Q', {*P_1 *, . . . ,* P_n*})* gelten soll. Dieser Pfad wird als *Context Path* bezeichnet.

Diese Definition wird nun für die Kontrollbeispiele 8.1 und 8.2 verwendet. Dafür soll die Schlüsselbedingung σ = (`uni.institut.vortragende.vt, (lva, {@titel})`) definiert werden. Mit dem Ansatz von Buneman et al. ist die Definition dieses Constraints möglich, da `uni.institut.vortragende.vt` als *Context Path Q* verwendet wird und `(lva, {@titel})` die eigentliche Schlüsselbedingung darstellt.

Wird dieser Constraint nun für den XML Baum in Abbildung 4.22 verwendet, ergeben sich für die Titel der Lehrveranstaltungen des ersten Vortragenden die Werte {*VO*} und {*UE*}. Die Lehrveranstaltungen können somit eindeutig identifiziert werden. Des Weiteren kann auch die Lehrveranstaltung mit dem Titel {*VO*} im Teilbaum des zweiten Vortragenden eindeutig identifiziert werden, da dies mit Hilfe von relativen Constraints unabhängig von den Daten des ersten Vortragenden möglich ist. Das Kontrollbeispiel 8.1 wird somit korrekt umgesetzt.

Als nächstes wird der Constraint anhand des XML Baums in Abbildung 4.23 getestet. Hier stellt sich heraus, dass eine eindeutige Identifizierung nicht mehr möglich ist, da beide Lehrveranstaltungen des ersten Vortragenden denselben Titel haben. Der Constraint ist somit – wie vorgesehen – nicht erfüllt und das Kontrollbeispiel 8.2 wird korrekt umgesetzt.

Das Vergleichsmerkmal 8 wird daher von diesem Ansatz erfüllt.

6.2.7 Nicht vorhandene Datenobjekte

Buneman et al. unterscheiden in ihrem Ansatz zwischen *Weak Keys* und *Strong Keys*, welche fehlende Datenobjekte unterschiedlich behandeln.

Damit ein *Strong Key* von einem XML Dokument erfüllt wird, müssen die identifizierenden Datenobjekte (d.h. alle *Field*-Knoten) für jeden *Selector*-Knoten genau einmal vorhanden sein. Fehlen ein oder mehrere *Field*-Knoten (egal ob bei einem oder mehreren *Selector*-Knoten), ist eine Erfüllung des Constraints nicht mehr gegeben.

Bei einem *Weak Key* ist es hingegen erlaubt, dass identifizierende Datenobjekte nicht vorhanden sind. Fehlen ein oder mehrere *Field*-Knoten eines *Selector*-Knotens, wird dieser von allen anderen *Selector*-Knoten im XML Dokument unterschieden. Somit werden zwei *Selector*-Knoten x und y immer dann als nicht gleich behandelt, wenn ein *Field*-

Knoten (erreichbar durch den Pfad P_i) für Knoten x oder Knoten y fehlt.

Die Definition von *Strong* und *Weak Keys* in diesem Ansatz ist nicht gleich zu setzen mit der Definition von *Strong* und *Weak Satisfaction* von Abschnitt 4.1.7. Wird ein *Strong Key* gemäß Buneman et al. definiert, ist es gar nicht möglich, dass dieser in einem nicht vollständigen XML Dokument erfüllt ist. Somit ist weder *Strong Satisfaction* noch *Weak Satisfaction* anwendbar. Die Schlüsselbedingungen in den Kontrollbeispielen 9.1, 9.2 und 9.3 können somit als *Strong Keys* nicht definiert werden.

Als *Weak Key* definiert ist es hingegen möglich, dass der Constraint für ein XML Dokument mit nicht vorhandenen Datenobjekten verwendet wird. Es werden dabei jene Knoten ignoriert, für die einer oder mehrere *Key Paths* fehlen. Wird dies nun anhand der Kontrollbeispiele 9.1, 9.2 und 9.3 und der dafür vorgesehenen Schlüsselbedingung σ = `(uni.vortragende.vt, {name, lva, inst})` getestet, führt dies zu folgenden Ergebnissen:

Unter der Annahme von *dne* ist der Schlüssel im Kontrollbeispiel 9.1 wie gefordert erfüllt, da die zweite Vortragende wegen fehlendem `inst`-Wert ignoriert wird und die erste Vortragende demnach eindeutig identifiziert werden kann.

Das Kontrollbeispiel 9.2 wird ebenfalls korrekt umgesetzt, weil die Schlüsselbedingung nur jene *Selector*-Knoten beim Vergleich berücksichtigt, deren *Field*-Knoten alle vorhanden sind. Demnach wird der Constraint nur für die erste Vortragende geprüft und ist daher erfüllt.

Wird die *Weak Key* Definition für das Kontrollbeispiel 9.3 verwendet, führt dies jedoch zu einem falschen Ergebnis. Hierbei werden die Daten beider Vortragenden wegen fehlender `inst`-Werte nicht berücksichtigt, sodass der Schlüssel für diesen XML Baum erfüllt ist. Dies ist jedoch für das Kontrollbeispiel nicht vorgesehen und es wird somit nicht korrekt umgesetzt. Das Vergleichsmerkmal 9.1 wird deshalb unter der *Weak Key* Definition nicht erfüllt.

Für das Vergleichsmerkmal 9.2 wird die Umsetzung von Constraint in XML Dokumenten mit nicht vorhandenen Datenobjekten unter der Annahme von *unk* getestet.

Der Schlüssel im Kontrollbeispiel 9.1 ist als *Weak Key* erfüllt, da wiederum nur die Daten der ersten Vortragenden zum Vergleich herangezogen werden. Das Kontrollbeispiel wird somit korrekt umgesetzt.

Nicht korrekt umgesetzt wird jedoch das Kontrollbeispiel 9.2. Hierbei wäre der Schlüssel

erfüllt, weil die Identifikation (wegen fehlendem `inst`-Wert der zweiten Vortragenden) lediglich für das erste Element `vt` geprüft wird und dieses somit eindeutig identifiziert werden kann. Dies ist im Kontrollbeispiel jedoch nicht gefordert.

Ebenfalls falsch umgesetzt wird das Kontrollbeispiel 9.3. Der Schlüssel wäre hier wiederum erfüllt, da die Daten beider Vortragenden wegen fehlender `inst`-Werte nicht berücksichtigt werden. Dies ist aber für dieses Kontrollbeispiel nicht vorgesehen.

Da die Kontrollbeispiele 9.2 und 9.3 nicht korrekt umgesetzt werden können, ist das Vergleichsmerkmal 9.2 für *Weak Keys* ebenfalls nicht erfüllt.

6.2.8 Mehrfach vorhandene Datenobjekte

Im Zusammenhang mit mehrfach vorhandenen Datenobjekten muss wiederum unterschieden werden zwischen *Strong Keys* und *Weak Keys*. Werden Schlüsselbedingungen in Form von *Strong Keys* verwendet, muss jeder *Field*-Knoten genau einmal vorhanden sein, sodass mehrfach vorhandene Datenobjekte nicht erlaubt sind. Somit wird das Vergleichsmerkmal 10 von *Strong Keys* nicht erfüllt.

Wird der Schlüssel als *Weak Key* definiert, sind mehrfach vorhandene Datenobjekte generell erlaubt. Buneman et al. stellen jedoch keine Methodik zur Verfügung, um eine semantisch korrekte Umsetzung von Schlüsseln in XML Bäumen mit mehrfach vorhandenen Datenobjekten zu gewährleisten. Vielmehr gehen sie davon aus, dass alle möglichen Kombinationen der *Field*-Knoten den *Selector*-Knoten eindeutig identifizieren können müssen. [51]

Angewendet auf das Kontrollbeispiel 10.1 würde dies bedeuten, dass für die erste LVA neben den richtigen Wertekombinationen {*DKE, Laura Leitner, WS12*} und {*DKE, Felix Schnell, WS11*} auch die falschen Wertemengen {*DKE, Laura Leitner, WS11*} und {*DKE, Felix Schnell, WS12*} gebildet werden. Dadurch würde die Wertekombination {*DKE, Laura Leitner, WS11*} sowohl bei der ersten als auch bei der zweiten LVA vorhanden sein und die Lehrveranstaltungen nicht mehr eindeutig identifiziert werden können. Eine korrekte Umsetzung des Kontrollbeispiels 10.1 ist somit nicht gewährleistet.

Das Kontrollbeispiel 10.2 wird ebenfalls nicht korrekt umgesetzt, da eine falsche Wertekombination bewirken würde, dass der Constraint erfüllt wäre. Würden nämlich für die erste LVA anstatt der richtigen Mengen {*DKE, Laura Leitner, WS12*} und {*DKE, Fe-*

lix Schnell WS11} die falschen Wertekombinationen {*DKE, Laura Leitner, WS11*} und {*DKE, Felix Schnell, WS12*} gebildet werden, wäre eine eindeutige Identifizierung möglich. Da dies für das Kontrollbeispiel nicht vorgesehen ist, wird es nicht korrekt umgesetzt.

Das Vergleichsmerkmal 10 ist somit auch unter der Definition von *Weak Keys* nicht erfüllt.

6.3 Zusammenfassung

Abbildung 6.1 fasst nochmals die Ergebnisse der Evaluation des Ansatzes von Buneman et al. zusammen. Das erste Vergleichsmerkmal ist vom Ansatz erfüllt, weil er an keine Schemasprache gebunden ist. Ebenfalls erfüllt ist das 2. Vergleichsmerkmal, da die Verwendung von Kleene Operatoren durch die Erweiterung der Pfadsprache in [19] ermöglicht wird.

Das von Buneman et al. verwendete Baummodell liefert mit der Funktion *ele(v)* eine Liste von Element- und Textknoten, welche in geordneter Reihenfolge aufgelistet sind. Des Weiteren ist das verwendete Baummodell angelehnt an das Baummodell DOM [56], welches das Vorhandensein von gemischtem Inhalt in Elementknoten erlaubt. Die Vergleichsmerkmale 3 und 4 sind somit erfüllt.

Das nächste Vergleichsmerkmal ist ebenfalls erfüllt, da durch die Verwendung einer Menge $\{P_1, \ldots, P_n\}$ von *Field*-Pfaden die Definition von n-ären Schlüsseln möglich ist.

Bei der Untersuchung der im Ansatz angegebenen Möglichkeiten des Wertvergleichs stellt sich heraus, dass das Vergleichsmerkmal 6 nicht erfüllt ist. Es wird zwar im Baummodell ermöglicht, dass gemischter Inhalt im XML Dokument vorkommt (siehe Vergleichsmerkmal 4), es wird jedoch keine Möglichkeit geboten, gemischten Inhalt beim Wertvergleich zu integrieren. Vielmehr wird ein baumbasierter Wertvergleich ermöglicht, sodass das Vergleichsmerkmal 7 vom Ansatz erfüllt wird.

Durch die Definition von relativen Schlüsseln als σ = *(Q, (Q', {*$P_1, \ldots, P_n$*}))* wird auch das 8. Vergleichsmerkmal erfüllt, da mit dem Pfad *Q* der Geltungsbereich der Constraints eingeschränkt werden kann.

Nicht erfüllt sind hingegen die beiden letzten Vergleichsmerkmale, welche sich mit Constraint in XML Dokumenten mit nicht oder mehrfach vorhandenen Datenobjekten beschäftigen. Als *Strong Keys* definiert, ist es generell nicht möglich, dass Datenobjekte nicht oder mehrfach vorhanden sind. Dies ist unter der *Weak Key* Definition jedoch erlaubt. Dennoch können die Kontrollbeispiele nicht korrekt umgesetzt werden und die Vergleichsmerkma-

le sind auch für *Weak Keys* nicht erfüllt.

Vergleichsmerkmale		
1	Schema	erfüllt
2	Pfadsprache	erfüllt
3	Baummodel: Dokumentordnung	erfüllt
4	Baummodell: Gemischter Inhalt	erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	erfüllt
8	Geltungsbereich	erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	nicht erfüllt

Abbildung 6.1: Vergleichsmerkmale für den Ansatz von Buneman et al.

Kapitel 7

Ansatz von Arenas und Libkin

Arenas und Libkin [6, 7] entwickelten einen Ansatz zur Definition von Funktionalen Abhängigkeiten für XML (XFDs), welcher im folgenden Kapitel erläutert wird.

7.1 Kurzbeschreibung

Arenas und Libkin definieren in ihrem Ansatz Funktionale Abhängigkeiten für XML, welche angelehnt an Funktionale Abhängigkeiten für relationale Datenbanken aufgebaut sind. Eine XFD wird dabei definiert als $\sigma = S_1 \rightarrow S_2$, wobei S_1 sowie S_2 jeweils für eine endliche, nicht-leere Menge von Pfaden in einem XML Dokument stehen.

Für die Umsetzung der XFDs werden die XML Dokumente, welche einer DTD entsprechen müssen, als ausgeflachte relationale Tupel abgebildet. Diese werden als *Tree-Tuples* bezeichnet. Die DTD erfüllt dabei die Aufgabe eines relationalen Schemas. Die verschiedenen Pfade zu den Blättern der DTD entsprechen den Attributen der Relation und die *Tree-Tuples* bilden die Tupel in der Relation. Jedem Pfad in S_1 bzw. S_2 wird ein Wert zugeordnet, welcher entweder der Knotenidentität von Elementknoten oder den Stringwerten von Attribut- und Textknoten entspricht.

Darüber hinaus wurde von Arenas und Libkin eine Normalform für XML Dokumente vorgestellt.

7.2 Überprüfung der Vergleichsmerkmale

Die in Kapitel 4 angegebenen Vergleichsmerkmale werden im folgenden Kapitel für den Ansatz von Arenas und Libkin überprüft.

7.2.1 Unabhängigkeit von einer Schemasprache

Zur Umsetzung der XFDs ist es notwendig, dass die XML Dokumente als Tupel abgebildet werden. Dabei wird davon ausgegangen, dass diese einer DTD entsprechen. Der Ansatz setzt somit voraus, dass eine DTD vorhanden ist. Das erste Vergleichsmerkmal ist daher nicht erfüllt.

7.2.2 Pfadsprache

Zur Definition der XFDs verwenden Arenas und Libkin einfache Abwärtspfade. Diese können auch leer sein. Ist dies nicht der Fall, müssen sie beim Wurzelknoten beginnen und in einem inneren Knoten, d.h. einem Elementknoten, oder einem Attribut- oder Textknoten enden. Es ist jedoch nicht möglich, Kleene Operatoren zur Definition der Constraints zu verwenden.

Für die Kontrollbeispiele 2.3 und 2.4 soll die Funktionale Abhängigkeit σ = (`_*.vt`, `{name}` $\rightarrow$ `{@id}`) definiert werden. Dazu muss ermöglicht werden, dass ein Pfad mit einem Elementknoten enden kann. Dies wird durch die Definition von Arenas und Libkin ermöglicht. Die zweite Anforderung ist, dass ein Kleene Operator verwendet werden kann. Dies wird von diesem Ansatz jedoch nicht ermöglicht, sodass beide Kontrollbeispiele nicht umgesetzt werden können. Aus diesem Grund ist das 2. Vergleichsmerkmal nicht erfüllt.

7.2.3 Baummodell

Im Ansatz von Arenas und Libkin wird nicht vorausgesetzt, dass ein XML Baum geordnet sein muss. Der XML Baum von Kontrollbeispiel 3.1 kann deshalb nicht abgebildet werden und das Vergleichsmerkmal 3 ist nicht erfüllt.

Des Weiteren verwenden Arenas und Libkin ein Baummodell, in dem ein Elementknoten beliebig viele Unterelemente beinhalten kann oder genau einen Textknoten. Dass sowohl Element- als auch Textknoten vorhanden sind ist nicht erlaubt, sodass gemischter Inhalt

nicht abgebildet werden kann. Das Kontrollbeispiel 4.1 kann daher nicht umgesetzt werden und das Vergleichsmerkmal 4 ist nicht erfüllt.

7.2.4 Arität

Arenas und Libkin definieren in ihrem Ansatz eine Funktionale Abhängigkeit als $\sigma = S_1 \rightarrow S_2$, wobei S_1 und S_2 jeweils für eine Menge von Pfaden steht. Es ist somit möglich, neben unären Constraints auch n-äre Constraints zu definieren.

Dies wird nun anhand der Kontrollbeispiele 5.3 und 5.4 getestet. Dafür soll die Funktionale Abhängigkeit σ = `(uni.vortragende.vt, {@vn, @nn}` $\rightarrow$ `{@id})` definiert werden, wobei `{@vn, @nn}` die Menge S_1 darstellt und `{@id}` bildet die Menge S_2.

Als nächstes wird für das Kontrollbeispiel 5.3 der XML Baum in Abbildung 4.9 als Tupel dargestellt, was zu folgendem Ergebnis führt[12]:

uni.vortragende.vt.@id	uni.vortragende.vt.@vn	uni.vortragende.vt.@nn
V1	Laura	Leitner
V2	Felix	Schnell

Da sich die Wertemengen {*Laura, Leitner*} und {*Felix, Schnell*} unterscheiden, können die `id`-Werte eindeutig bestimmt werden. Das Kontrollbeispiel 5.3 wird somit korrekt umgesetzt.

Wird die Funktionale Abhängigkeit nun am XML Baum in Abbildung 4.10 getestet, ergeben sich folgende Tupel:

uni.vortragende.vt.@id	uni.vortragende.vt.@vn	uni.vortragende.vt.@nn
V1	Laura	Leitner
V2	Laura	Leitner

Hier wird ersichtlich, dass die `id`-Elemente nicht eindeutig bestimmt werden können, weil die Wertemenge{*Laura, Leitner*} zweimal vorhanden ist. Der Constraint ist für diesen XML Baum daher nicht erfüllt. Dies ist im Kontrollbeispiel 5.4 gefordert, sodass es ebenfalls korrekt umgesetzt werden kann. Das Vergleichsmerkmal 5 wird somit erfüllt.

[12]Es werden im Folgenden nur die für den Constraint relevanten Tupel dargestellt.

7.2.5 Wertvergleich

Im Ansatz von Arenas und Libkin werden Elementknoten anhand ihrer Identität verglichen. Zwei Elementknoten sind somit gleich, wenn sie die gleiche Identität haben. Eine eigene Definition der Wertgleichheit zweier Elementknoten ist nicht vorhanden.

Um das Vergleichsmerkmal 6 zu erfüllen, soll ein Wertvergleich basierend auf der Verkettung von Textwerten ermöglicht werden. Dies wird im Ansatz von Arenas und Libkin nicht ermöglicht, da weder gemischter Inhalt in einem XML Dokument vorhanden sein darf noch eine Definition der Wertgleichheit angegeben wird. Das Vergleichsmerkmal 6 ist deshalb nicht erfüllt.

Für das Vergleichsmerkmal 7 soll hingegen ein baumbasierter Wertvergleich verwendet werden können. Auch dies wird im Ansatz nicht unterstützt. Dennoch ist es möglich, das im Kontrollbeispiel 7.3 gewünschte Ergebnis zu erzielen. Die Funktionale Abhängigkeit σ = (`uni.vortragende.vt`, {`personendaten`} $\rightarrow$ {`@id`}) wird dabei anstatt mit Wertvergleich mit Identitätsvergleich (Knoten-Vergleich) validiert. Die beiden Elemente `personendaten` haben verschiedene Identitäten, sodass Sie voneinander unterschieden und die Elemente `vt` eindeutig bestimmt werden können.

Wird die Funktionale Abhängigkeit jedoch gemäß Kontrollbeispiel 7.4 für den XML Baum in Abbildung 4.19 verwendet, stellt sich heraus, dass eine korrekte Umsetzung nicht möglich ist. Die beiden `personendaten`-Elemente haben wiederum eine unterschiedliche Identität, sodass die `vt`-Elemente eindeutig bestimmt werden könnten. Dies ist im Kontrollbeispiel 7.4 jedoch nicht gefordert, da die Elemente `personendaten` der beide Vortragenden sowohl dieselbe Struktur als auch dieselben Werte haben, sodass die Vortragenden nicht eindeutig bestimmt werden können. Eine korrekte Umsetzung des Kontrollbeispiels ist somit nicht gegeben und das Vergleichsmerkmal 7 wird nicht erfüllt.

7.2.6 Geltungsbereich von Constraints

Arenas und Libkin bieten in ihrem Ansatz nur die Möglichkeit, Funktionale Abhängigkeiten zu definieren, deren Pfade direkt am Wurzelknoten beginnen. Eine Definition von relativen Pfaden ist nicht möglich. Daher können die Kontrollbeispielen 8.3 und 8.4 nicht umgesetzt werden und das Vergleichsmerkmal 8 ist nicht erfüllt.

7.2.7 Nicht vorhandene Datenobjekte

Sind Datenobjekte in einem XML Baum nicht vorhanden, werden diese im Ansatz von Arenas und Libkin mit dem Symbol $\perp$ dargestellt. Ein Null-Knoten ist zu allen anderen Knoten unterschiedlich. Um dies zu ermöglichen wird die Kennzeichnung $\perp_1, \perp_2, \perp_3, ...$ verwendet. Generell wird also eine Methode geboten, um mit nicht vorhandenen Datenobjekten umzugehen.

Es soll nun mit Hilfe der Kontrollbeispiele 9.4, 9.5 und 9.6 ermittelt werden, ob Constraints mit nicht vorhandenen Datenobjekten auch semantisch korrekt umgesetzt werden können. Dazu wird die Funktionale Abhängigkeit σ = (`uni.vortragende.vt, {name.S, lva.S, inst.S}` $\rightarrow$ `{@id}`) verwendet.

Im Kontrollbeispiel 9.4 wird gefordert, dass die `id`-Attribute der Vortragenden im XML Baum in Abbildung 4.25 sowohl unter der Annahme *dne* als auch *unk* eindeutig bestimmt werden. Es ergeben sich folgende Tupel:

uni.vortragende.vt

@id	name.S	lva.S	inst.S
V1	Laura Leitner	VO DKE	DKE
V2	Laura Leitner	UE DKE	$\perp_1$

Durch die Verwendung des Symbols $\perp_1$ für das fehlende Datenobjekt `inst` der zweiten Vortragenden kann die ID der Vortragenden eindeutig bestimmt werden. Das Kontrollbeispiel wird somit sowohl unter der Annahme von *dne* als auch *unk* semantisch korrekt umgesetzt.

Für das Kontrollbeispiel 9.5 wird der XML Baum in Abbildung 4.26 verwendet. Hierbei haben zwei Vortragende die gleichen Werte für die Elemente `name` und `lva`, das Element `inst` fehlt jedoch bei der zweiten Vortragenden. Da das fehlende Datenobjekt mit dem Symbol $\perp_1$ abgebildet wird, ergeben sich folgende Tupel:

uni.vortragende.vt

@id	name.S	lva.S	inst.S
V1	Laura Leitner	UE DKE	DKE
V2	Laura Leitner	UE DKE	$\perp_1$

Da sich das Symbol $\perp_1$ von allen anderen Werten unterscheidet, kann die ID der Vortragenden eindeutig bestimmt werden. Unter der Annahme von *dne* ist dies gefordert, sodass

das Kontrollbeispiel semantisch korrekt umgesetzt wird. Für *unk* sollte jedoch ein anderes Ergebnis erzielt werde. Das Kontrollbeispiel 9.5 ist daher unter der Annahmen von *unk* nicht erfüllt.

Wird nun die Funktionale Abhängigkeit für den XML Baum in Abbildung 4.27 verwendet, ergeben sich folgende Tupel:

uni.vortragende.vt

@id	name.S	lva.S	inst.S
V1	Laura Leitner	UE DKE	$\perp_1$
V2	Laura Leitner	UE DKE	$\perp_2$

Beide Vortragende haben dieselben Werte für die Elemente `name` und `lva`, für das Element `inst` sind jedoch keine Werte vorhanden. Da im Ansatz von Arenas und Libkin nummerierte $\perp$-Symbole verwendet werden können (im Beispiel $\perp_1$ und $\perp_2$) und sich diese unterscheiden, können die IDs beider Vortragenden eindeutig bestimmt werden. Da dies weder unter der Annahme von *dne* noch *unk* gefordert ist, wird das Kontrollbeispiel 9.6 für beide Fälle nicht korrekt umgesetzt.

Da weder für das Vergleichsmerkmal 9.1 noch für das Vergleichsmerkmal 9.2 alle Kontrollbeispiele korrekt umgesetzt werden können, sind beide Vergleichsmerkmale nicht erfüllt.

7.2.8 Mehrfach vorhandene Datenobjekte

Bei diesem Ansatz werden die XML Bäume ausgeflacht und als flache Relationen abgebildet. Die Attribute der daraus resultierenden Tupel entsprechen den in der zugehörigen DTD angegebenen Pfaden. Sind nun in einem Unterbaum bestimmte Datenobjekte mehrfach vorhanden, wird für jedes dieser Datenobjekte ein eigener Tupel angeführt. Dadurch ist garantiert, dass nur jene Datenobjekte zu einem Tupel zusammengefasst werden, deren Zusammengehörigkeit semantisch korrekt ist.

Um dies zu kontrollieren wird die Funktionale Abhängigkeit σ = `(uni.lvas.lva, {titel.S, vt.name.S, vt.sem.S}` $\rightarrow$ `{@id})` anhand der Kontrollbeispiele 10.3 und 10.4 getestet. Es wird dabei angenommen, dass die verwendeten XML Dokumente der DTD in Abbildung 7.1 entsprechen[13].

[13] Zur besseren Übersicht wird hier nur die DTD für den Teilbaum `lvas` angegeben.

```
<?xml encoding="UTF-8"?>
<!ELEMENT uni (lvas)>
<!ELEMENT lvas (lva*)>
<!ELEMENT lva (titel, vt+)>
<!ATTLIST lva id CDATA #REQUIRED>
<!ELEMENT titel (#PCDATA)>
<!ELEMENT vt (name, sem)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sem (#PCDATA)>
```

Abbildung 7.1: DTD zur Überprüfung der Kontrollbeispiele 10.3 und 10.4

Für das Kontrollbeispiel 10.3 wird der Teilbaum `lvas` aus Abbildung 4.29 folgendermaßen entschachtelt:

@id	titel.S	vt.name.S	vt.sem.S
V1	DKE	Laura Leitner	WS12
V1	DKE	Felix Schnell	WS11
V2	DKE	Laura Leitner	WS11

Das erste Element `lva` hat zwei Unterelemente `vt` mit jeweils verschiedenen Werten für das Unterelement `name`. Für jedes dieser Unterelemente wird ein eigener Tupel angelegt und somit die semantisch korrekten Kombinationen der Werte gebildet. Dadurch wird ermöglicht, dass die IDs der beiden Lehrveranstaltungen trotz mehrfach vorhandener `name`-Elemente eindeutig bestimmt werden können. Das Kontrollbeispiel 10.3 wird somit semantisch korrekt umgesetzt.

Wird nun der Teilbaum `lvas` aus Abbildung 4.30 als Relation abgebildet, resultieren folgende Tupel:

@id	titel.S	vt.name.S	vt.sem.S
V1	DKE	Laura Leitner	WS12
V1	DKE	Felix Schnell	WS11
V2	DKE	Felix Schnell	WS11

Hierbei ergeben sich beim zweiten und dritten Tupel dieselben Werte für die Attribute `titel.S`, `vt.name.S` und `vt.sem.S`, jedoch unterschiedliche Werte für das Attribut

`@id`. Der Constraint ist somit nicht erfüllt. Dies ist für das Kontrollbeispiel 10.4 vorgesehen, sodass es korrekt umgesetzt werden kann. Das Vergleichsmerkmal 10 wird daher von diesem Ansatz erfüllt.

7.3 Zusammenfassung

Die Auflistung in Abbildung 7.2 fasst die Ergebnisse der Analyse dieses Ansatzes nochmals zusammen.

Das erste Vergleichsmerkmal ist nicht erfüllt, da im Ansatz die XML Dokumente entsprechend einer DTD als Tupel abgebildet werden. Somit ist der Ansatz an die Schemasprache DTD gebunden. Ebenso nicht erfüllt ist das zweite Vergleichsmerkmal, da zwar erlaubt ist, dass Pfade in Constraints in Elementknoten enden, die Verwendung eines Kleene Operators jedoch nicht unterstützt wird.

Die dem Baummodell zugeordneten Vergleichsmerkmale werden ebenfalls nicht erfüllt. Das Vergleichsmerkmal 3 fordert das Einhalten der Dokumentordnung, was durch die Abbildung der XML Dokumente als ungeordnete Tupel nicht möglich ist. Um das Vergleichsmerkmal 4 zu erfüllen, müsste die Verwendung von gemischtem Inhalt in XML Dokumenten unterstützt werden. Da dies in diesem Ansatz nicht der Fall ist, wird auch dieses Vergleichsmerkmal nicht erfüllt.

Die Arität betreffend stellt sich heraus, dass das Vergleichsmerkmal 5 erfüllt ist. Arenas und Libkin definieren ihre XFD als $\sigma = S_1 \rightarrow S_2$, wobei S_1 und S_2 jeweils Mengen von Pfaden darstellen. Die Definition von n-ären Constraints wird somit ermöglicht.

Zum Vergleich zweier Elementknoten aufgrund ihrer Werte wurde im Ansatz keine Definition angegeben. Vielmehr werden Knoten aufgrund ihrer Identität unterschieden. Dadurch werden weder der Wertvergleich basierend auf der Verkettung von Textwerten noch der baumbasierte Wertvergleich unterstützt. Die Vergleichsmerkmale 6 und 7 sind somit nicht erfüllt. Ebenfalls nicht erfüllt ist das Vergleichsmerkmal 8, da die Definition von relativen Constraints im Ansatz nicht ermöglicht wird.

Der Ansatz bietet eine Möglichkeit, XFDs für unvollständige XML Dokumente zu definieren. Nicht vorhandene Datenobjekten werden dabei mit dem Symbol $\perp$ ersetzt. Diese Methode führt jedoch nicht immer zu dem gewünschten Ergebnis, sodass weder Vergleichsmerkmal 9.1 noch Vergleichsmerkmal 9.2 erfüllt sind. Im Gegensatz dazu wird durch die Aufspaltung der XML Bäume in Tupel garantiert, dass mehrfach vorhandene

Vergleichsmerkmale		
1	Schema	nicht erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	nicht erfüllt
4	Baummodell: Gemischter Inhalt	nicht erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	nicht erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	erfüllt

Abbildung 7.2: Vergleichsmerkmale für den Ansatz von Arenas und Libkin

Datenobjekte korrekt kombiniert werden. Dadurch wird die Definition von Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten vom Ansatz unterstützt und das Vergleichsmerkmal 10 ist erfüllt.

Kapitel 8

Ansatz von Fan und Libkin

Fan und Libkin [34] entwickelten einen Ansatz zur Definition von Schlüsseln, Fremdschlüsseln und Inklusionsabhängigkeiten in XML. Dieser wird im Folgenden vorgestellt.

8.1 Kurzbeschreibung

Fan und Libkin definieren einen Schlüssel als $\sigma = \tau[X] \rightarrow \tau$, wobei τ für einen Elementtyp einer DTD steht und X repräsentiert eine Menge von Attributen von τ.

Inklusionsabhängigkeiten werden definiert als $\sigma = \tau_1[X] \subseteq \tau_2[Y]$ mit τ_1 und τ_2 als Elementtypen und X bzw. Y als nicht leere Listen von Attributen, welche wiederum Attribute von τ_1 bzw. τ_2 sein müssen.

Fremdschlüsseln werden von Fan und Libkin mittels Kombination von einer Inklusionsabhängigkeit und einem Schlüssel dargestellt, d.h. die Inklusionsabhängigkeit $\sigma = \tau_1[X] \subseteq \tau_2[Y]$ wird kombiniert mit dem Schlüssel $\sigma = \tau_2[Y] \rightarrow \tau_2$. Die RHS der Inklusionsabhängigkeit referenziert also auf die Schlüsselbedingung.

Neben der Definition oben genannter Integritätsbedingungen gehen Fan und Libkin auch auf das Konsistenzproblem ein, welches sich mit der Frage beschäftigt, ob für eine angegebene Menge von XML Constraints zumindest ein XML Dokument existiert, dass diese Constraints erfüllt [33]. Des Weiteren beschreiben sie, wie sich DTDs und Integritätsbedingungen gegenseitig (negativ) beeinflussen können.

8.2 Überprüfung der Vergleichsmerkmale

Im Folgenden wird der Ansatz anhand der im Kapitel 4 vorgestellten Vergleichsmerkmale analysiert.

8.2.1 Unabhängigkeit von einer Schemasprache

Der Ansatz von Fan und Libkin ist an die Schemasprache DTD gebunden, d.h. es muss eine DTD für das XML Dokument vorhanden sein. Das 1. Vergleichsmerkmal ist somit nicht erfüllt.

8.2.2 Pfadsprache

In diesem Ansatz wird keine Pfadsprache definiert. Vielmehr wird auf die Knoten mit Hilfe der vorhandenen DTD zugegriffen. Eine DTD wird dabei folgendermaßen definiert:

Definition 8.1: *"A DTD (Document Type Definition) is defined to be D = (E, A, P, R, r), where*

- *E is a finite set of element types;*
- *A is a finite set of attributes, disjoint from E;*
- *P is a mapping from E to element type definitions: for each $\tau \in E, P(\tau)$ is a regular expression α defined as followes:*

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

  where S denotes string type, $\tau' \in E$, ϵ is the empty word, and "|", "," and "" denote union, concatenation, and the Kleene closure, respectively;*
- *R is a mapping from E to $\mathcal{P}(A)$, the power-set of A; if $l \in R(\tau)$, then we say l is defined for τ;*
- *$r \in E$ and is called the element type of the root."*

Wird nun für die XML Bäume in den Abbildungen 4.2 und 4.3 die dazugehörige DTD gemäß der Definition von Fan und Libkin definiert, führt dies zu folgendem Ergebnis:

```
E = {uni, inst, lvas, vortragende, lva, vt, name}
A = {titel, id}
```

```
P(uni) = inst, lvas, vortragende
P(inst) = lva*
P(lvas) = lva*
P(vortragende) = vt*
P(lva) = vt
P(vt) = name
P(name) = S
R(lva) = titel
R(vt) = id
r= uni
```

Für die Kontrollbeispiele sollen nun der Schlüssel σ = (_*.vt, {name}) und die Inklusionsabhängigkeit σ = (_*.lva.vt, [name]) $\subseteq$ (_*.vortragende.vt, [name]) definiert werden. Dies wird von diesem Ansatz leider nicht ermöglicht, weil als *Fields* in der Schlüsselbedingung immer eine Menge von Attributen verwendet werden müssen. Bei diesem Schlüssel müsste jedoch ein Elementtyp als *Field* verwendet werden. Da dies mit diesem Ansatz nicht ermöglicht wird, kann der Schlüssel nicht definiert werden. Selbiges gilt für die Inklusionsabhängigkeit.

Da keine Pfadsprache definiert wurde, wird die Verwendung von Kleene Operatoren auch nicht unterstützt. Dies kann jedoch mit der Definition der Constraints auf der Grundlage einer DTD umgangen werden. Würde das Element name als Attribut definiert werden mit R (vt) = id, name, so würde der Schlüssel lauten: σ = vt [name] $\rightarrow$ vt. Dadurch kann jedes Element vt, egal an welcher Stelle es sich im XML Baum befindet, eindeutig mit dessen Attribut name identifiziert werden. Dies würde für die Kontrollbeispiele 2.1 und 2.2 zum richtigen Ergebnis führen.

Die Inklusionsabhängigkeit und der Fremdschlüssel könnten selbst mit dieser Vorgehensweise nicht umgesetzt werden. Die Inklusionsabhängigkeit kann nicht definiert werden, da X bzw. Y Attribute des Elements vt sind, jedoch nicht unterschieden werden kann, ob es sich um ein Element vt im Teilbaum lva oder vortragende handelt.

Da die Kontrollbeispiele 2.1 und 2.2 bzw. 2.5 und 2.6 nicht korrekt umgesetzt werden können, ist das Vergleichsmerkmal 2 nicht erfüllt.

8.2.3 Baummodell

Fan und Libkin definieren einen Baum folgendermaßen:

Definition 8.2: *"Let D = (E, A, P, R, r) be a DTD. An XML Tree T valid with respect to D (conforming to D) is defined to be T = (V, lab, ele, att, val, root), where*

- *V is a finite set of nodes (vertices);*
- *lab is a function that maps each node in V to a label in E ∪ A ∪ {S}; a node v ∈ V is called an element of τ if lab(v) = τ and τ ∈ E, an attribute if lab(v) ∈ A, and a text node if lab(v) = S;*
- *ele is a partial function defined on elements in V; for any τ ∈ E, it maps each element v of type τ to a (possibly empty) list $[v_1, \cdots, v_n]$ of elements and text nodes in V such that $lab(v_1) \cdots lab(v_n)$ is in the regular language defined by $P(\tau)$;*
- *att is a partial function from V × A to V such that for any v ∈ V and l ∈ A, att(v, l) is defined iff lab(v) = τ, τ ∈ E and l ∈ R(τ);*
- *val is a partial function from V to string values such that for any node v ∈ V, val(v) is defined iff lab(v) = S or lab(v) ∈ A;*
- *root is the unique node in V such that lab(root) = r, called the root of T."*

Mit der Funktion *ele(v)* wird somit eine Liste geordneter Elemente und Textknoten des Knotens *v* zurückgegeben, wobei die Dokumentordnung eingehalten wird. Der XML Baum in Abbildung 4.6 kann daher abgebildet werden und das Vergleichsmerkmal 3 ist erfüllt.

Des Weiteren ist gemischter Inhalt erlaubt, da mit *ele(v)* sowohl Elemente als auch Textknoten zurückgegeben werden, welche wiederum geordnet sind. Der XML Baum in Abbildung 4.8 kann somit abgebildet werden und das Vergleichsmerkmal 4 ist erfüllt.

8.2.4 Arität

Wie bereits erwähnt wurde, definieren Arenas und Libkin Schlüsselbedingungen als $\sigma = \tau[X] \rightarrow \tau$, wobei *X* für eine Menge von Attributen steht. Die Definition von n-ären Constraints wird daher ermöglicht.

Dies wird nun anhand der Kontrollbeispiele getestet. Für das Kontrollbeispiel 5.1 soll der Schlüssel σ = `(uni.vortragende.vt, {@vn, @nn}`) definiert werden. Mit dem Ansatz von Arenas und Libkin wird dies folgendermaßen umgesetzt: $\sigma =$ `vt [@vn, @nn]` $\rightarrow$ `vt`. Dies führt für das Kontrollbeispiel 5.1 zum richtigen Ergebnis, da der Constraint für den XML Baum in Abbildung 4.9 erfüllt ist. Ebenfalls richtig wird das Kontrollbeispiel 5.2 umgesetzt, da der Constraint hier, wie gefordert, nicht erfüllt ist.

Für Schlüsselbedingungen ist das Vergleichsmerkmal 5 somit erfüllt.

Als nächstes wird die Definition von Inklusionsabhängigkeiten und Fremdschlüsseln getestet. Inklusionsabhängigkeiten werden definiert als $\sigma = \tau_1[X] \subseteq \tau_2[Y]$, wobei X bzw. Y wiederum für eine Menge von Attributen steht. Daher wird die Definition der n-ären Constraints ebenfalls ermöglicht.

Die Umsetzung wird nun anhand der Kontrollbeispiele 5.5 und 5.6 und dem Constraint σ = `(uni.lvas.lva.vt, [@vn, @nn])` $\subseteq$ `(uni.vortragende.vt, [@vn, @nn])` getestet. Hierzu muss der XML Baum in Abbildung 4.9 etwas verändert werden, da aufgrund der fehlenden Pfadsprache der Zugriff auf die Daten mittels DTD erfolgt und dabei nicht definiert werden kann, wo im XML Baum sich das Element `vt` befindet. Aus diesem Grund wird zum Testen dieses Kontrollbeispiels das Element `vt` im Teilbaum `lvas` umbenannt in `dozent`.

Der Fremdschlüssel ergibt sich nun aus der Inklusionsabhängigkeit σ =`dozent [@vn, @nn]` $\subseteq$ `vt [@vn, @nn]` und dem Schlüssel σ = `vt [@vn, @nn]`. Wird dies gemäß Kontrollbeispiel 5.5 und XML Baum in Abbildung 4.9 getestet, ergibt sich ein positives Ergebnis. Der Constraint ist wie gewünscht erfüllt. Angewendet auf den XML Baum in Abbildung 4.10 ist der Constraint – wie gefordert – nicht erfüllt.

Da alle Kontrollbeispiele korrekt umgesetzt werden können, ist das Vergleichsmerkmal 5 somit auch für Inklusionsabhängigkeiten und Fremdschlüssel erfüllt.

8.2.5 Wertvergleich

Mit der Funktion *val(v)* des Baummodells wird für alle Attribut- und Textknoten deren Stringwert zurückgegeben. Für Elementknoten ist *val(v)* nicht definiert. Sie werden anhand ihrer Identität verglichen. Der Wertvergleich ist somit auf einen reinen Stringvergleich beschränkt. Dadurch werden weder ein Wertvergleich basierend auf der Verkettung von Textwerten noch der baumbasierte Wertvergleich ermöglicht. Darüber hinaus beschränken Arenas und Libkin die *Field*-Knoten der Constraints auf Attributknoten.

Die Umsetzung der Kontrollbeispiele 6.1 und 6.2 bzw. 6.5 und 6.6 ist daher nicht möglich und das Vergleichsmerkmal 6 ist nicht erfüllt.

Selbiges gilt für den baumbasierten Wertvergleich und die Kontrollbeispiele 7.1 und 7.2 bzw. 7.5 und 7.6. Vergleichsmerkmal 7 ist somit ebenfalls nicht erfüllt.

8.2.6 Geltungsbereich von Constraints

Im Ansatz von Fan und Libkin ist eine Einschränkung des Geltungsbereichs von Constraints nicht möglich. Dies wird durch die Kontrollbeispiele 8.1 und 8.2 bzw. 8.5 und 8.6 verdeutlicht.

Hierzu sollte der Schlüssel σ = `(uni.institut.vortragende.vt, (lva, {@titel}))` definiert werden. Dies wird jedoch nicht ermöglicht, da der Schlüssel lediglich definiert werden kann als σ = `lva [@titel]` $\rightarrow$ `lva`. Dies führt für das Kontrollbeispiel 8.1 nicht zum richtigen Ergebnis, da der `@titel`-Wert der ersten und letzten Lehrveranstaltung gleich ist und eine eindeutige Identifizierung nur mit relativen Constraints möglich ist. Richtig umgesetzt wird hingegen Kontrollbeispiel 8.2, da auch die Definition von absoluten Constraints zum richtigen Ergebnis führt.

Für Inklusionsabhängigkeiten und Fremdschlüssel ergibt sich dasselbe Ergebnis. Kontrollbeispiel 8.5 ist mittels absoluten Constraints nicht erfüllbar, sodass es von diesem Ansatz nicht korrekt umgesetzt werden kann. Kontrollbeispiel 8.6 wird jedoch auch ohne Einschränkung des Geltungsbereich richtig umgesetzt.

Da die Kontrollbeispiele 8.1 und 8.5 nicht korrekt umgesetzt werden können, ist das Vergleichsmerkmal 8 weder für Schlüssel noch für Inklusionsabhängigkeiten und Fremdschlüssel erfüllt.

8.2.7 Nicht vorhandene Datenobjekte

Fan und Libkin fordern in ihrem Ansatz, dass alle *Field*-Knoten für jeden *Selector*-Knoten vorhanden sein müssen. Dies gilt für Schlüssel, Inklusionsabhängigkeiten und Fremdschlüssel. Dadurch ist es nicht möglich, Constraints für XML Dokumente zu definieren, in denen Datenobjekte fehlen. Aufgrund dieser Einschränkung sind die Kontrollbeispiele nicht umsetzbar und die Vergleichsmerkmale 9.1 und 9.2 nicht erfüllt.

8.2.8 Mehrfach vorhandene Datenobjekte

In diesem Ansatz sind die *Fields* auf Attribute (des *Selector*-Elements) beschränkt. Da es jedoch nicht erlaubt ist, dass ein Elementknoten mehrere Attributknoten mit demselben Label beinhaltet (siehe Kapitel 2), ist der Umgang mit XML Dokumenten mit mehrfach vorhandenen Datenobjekten nicht möglich. Die Kontrollbeispiele können daher nicht umgesetzt werden und das Vergleichsmerkmal 10 ist nicht erfüllt.

Vergleichsmerkmale		
1	Schema	nicht erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	erfüllt
4	Baummodell: Gemischter Inhalt	erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	nicht erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	nicht erfüllt

Abbildung 8.1: Vergleichsmerkmale für den Ansatz von Fan und Libkin

8.3 Zusammenfassung

Zusammenfassend ergibt sich für diesen Ansatz das in Abbildung 8.1 dargestellte Ergebnis. Der Ansatz basiert auf der Schemasprache DTD, sodass das erste Vergleichsmerkmal nicht erfüllt ist. Des Weiteren wird keine Pfadsprache angegeben, sodass das Vergleichsmerkmal 2 ebenfalls nicht erfüllt ist. Im Gegensatz dazu sind die Vergleichsmerkmale 3 und 4 erfüllt, da durch die Definition der Funktion *ele(v)* im Baummodell das Einhalten der Reihenfolge garantiert ist und gemischter Inhalt im XML Dokument unterstützt wird. Ebenfalls erfüllt ist das 5. Vergleichsmerkmal, da n-äre Constraints definiert werden können.

Der Wertvergleich wird von Fan und Libkin auf einfachen String-Vergleich beschränkt, sodass die Vergleichsmerkmale 6 (Wertvergleich durch Verkettung von Textwerten) und 7 (Baumbasierter Wertvergleich) nicht erfüllt sind. Des Weiteren ist keine Einschränkung des Geltungsbereichs möglich, sodass das Vergleichsmerkmal 8 ebenfalls nicht erfüllt ist.

Da die *Field*-Knoten genau einmal für jeden *Selector*-Knoten vorhanden sein müssen, sind auch die Vergleichsmerkmale 9.1 und 9.2 sowie 10 nicht erfüllt.

Kapitel 9

Ansatz von Lee, Ling, Low

In diesem Abschnitt wird die Definition von Funktionalen Abhängigkeiten für XML von Lee, Ling und Low [58] analysiert.

9.1 Kurzbeschreibung

Eine Funktionale Abhängigkeit wird von Lee, Ling und Low definiert als $\sigma = (Q, [P_{x_1}, \cdots, P_{x_n} \rightarrow P_y])$. Dabei entspricht Q dem sogenannten *Header*, mit dem der Geltungsbereich der Funktionale Abhängigkeit definiert wird. $P_{x_1}, \cdots, P_{x_n}$ entspricht der LHS und P_y der RHS der Funktionalen Abhängigkeit. Hierbei steht $P_{x_i}, 1 \leq i \leq n$, für einen Entitätstyp, der aus einem Elementnamen und optionalen Schlüsselattribut(en) besteht, und P_y für einen Entitätstyp, der aus einem Elementnamen und einem optionalen Attributnamen besteht.

Bei der Definition der Constraints werden jedoch Einschränkungen angegeben. Generell unterscheiden Lee, Ling und Low zwischen wohlstrukturierten und nicht wohlstrukturierten Funktionalen Abhängigkeiten. Damit ein Constraint wohlstrukturiert ist, müssen die Elemente und Entitätstypen einer sogenannten *lineage* (kann informell übersetzt werden mit *Abstammungslinie*) entsprechen. Dies bedeutet, dass die Elemente in *Q*, die Entitätstypen der LHS und die Entitätstypen der RHS eine *lineage* bilden müssen. Eine *lineage* kann mit einem Abwärtspfad verglichen werden.

Die Funktionale Abhängigkeit σ = (/uni/lvas/lva, [vt.@name $\rightarrow$ vid.@id]) ist wohlstrukturiert, da die Elemente bzw. Entitätstypen {uni, lvas, lva, vt, vid} eine *lineage* bilden. Nicht wohlstrukturiert ist hingegen die Funktionale Abhängigkeit σ =

(/uni/lvas/lva/vt, [name → id]), wenn name kein Vorfahren-Knoten von id ist, sodass {uni, lvas, lva, vt, name, id} keine *lineage* bildet.

9.2 Überprüfung der Vergleichsmerkmale

Im Folgenden werden die in Kapitel 4 vorgestellten Vergleichsmerkmale überprüft und anschließend zusammengefasst.

9.2.1 Unabhängigkeit von einer Schemasprache

Lee, Ling und Low beschreiben den Aufbau einer Funktionalen Abhängigkeit mit Hilfe einer DTD. Diese ist folgendermaßen definiert:

```
1 <!ELEMENT Constraints (Fd*)>
2 <!ELEMENT Fd (HeaderPath,LHS+,RHS)>
3 <!ATTLIST Fd Fid ID #REQUIRED>
4 <!ELEMENT LHS (ElementName,Attribute*)>
5 <!ELEMENT RHS (ElementName,Attribute*)>
6 <!ELEMENT HeaderPath (#PCDATA)>
7 <!ELEMENT ElementName (#PCDATA)>
8 <!ELEMENT Attribute (#PCDATA)>
```

Abbildung 9.1: Aufbau einer Funktionalen Abhängigkeit als DTD dargestellt

Die Autoren gehen zwar bei ihrer Grunddefinition von Funktionalen Abhängigkeiten nicht von dem Vorhandensein einer DTD aus, empfehlen dies jedoch. Ihre erweiterten Definitionen setzen jedoch voraus, dass eine DTD vorhanden ist. Das Vergleichsmerkmal 1 ist somit nicht erfüllt.

9.2.2 Pfadsprache

In diesem Ansatz wird für den Header Q eine *fully qualified path expression* verwendet. Dies bedeutet, dass der Pfad bei einem Wurzelknoten beginnen muss. Die Verwendung eines Kleene Operators ist somit nicht möglich, sodass die Funktionale Abhängigkeit σ = (_*.vt, {name} → {@id}) der Kontrollbeispiele 2.3 und 2.4 nicht definiert werden kann. Das zweite Vergleichsmerkmal ist somit nicht erfüllt.

9.2.3 Baummodell

Lee, Ling und Low gehen in ihrem Ansatz von der vom W3C [13] definierten Dokumentordnung aus. Die Knoten ein einem XML Dokument müssen demnach einer Ordnung unterliegen. Der XML Baum in Abbildung 4.6 kann daher abgebildet werden und das Vergleichsmerkmal 3 ist somit erfüllt.

Des Weiteren wird in der XML Spezifikation [17] vom W3C das Vorhandensein von gemischtem Inhalt unterstützt. Aus diesem Grund erfüllt der Ansatz auch das Vergleichsmerkmal 4, da der XML Baum in Abbildung 4.8 ebenfalls abgebildet werden kann.

9.2.4 Arität

Aus der Definition $\sigma = (Q, [P_{x_1}, \cdots, P_{x_n} \rightarrow P_y])$ geht bereits hervor, dass mit $P_{x_1}, \cdots, P_{x_n}$ auf der LHS n-äre Constraints ausgedrückt werden können.

Für die Kontrollbeispiele 5.3 und 5.4 soll die Funktionale Abhängigkeit σ = `(uni.vortragende.vt, {@vn, @nn}` $\rightarrow$ `{@id})` definiert werden. Da hierbei keine *lineage* vorhanden ist, kann kein wohlstrukturierter Constraint gemäß diesem Ansatz definiert werden.

Lee, Ling und Low beschreiben jedoch in ihrem Ansatz eine Klasse von Funktionalen Abhängigkeiten, die auch ohne Einhaltung der *lineage* definiert werden können. Hierbei handelt es sich um Constraints in flachen XML Bäumen, d.h. die Elemente sind wenig oder gar nicht verschachtelt und die Daten werden in Form von Attributen gespeichert.

Zum Testen der Kontrollbeispiele 5.3 und 5.4 wäre es nun möglich, die XML Bäume in den Abbildungen 4.9 und 4.10 leicht einzugrenzen und den Elementknoten `vortragende` als Wurzelknoten zu verwenden. Die Funktionale Abhängigkeit würde demnach lauten σ = `(/vortragende/vt, [@vn, @nn` $\rightarrow$ `@id])`. Dies führt für das Kontrollbeispiel 5.3 zum richtigen Ergebnis, da die IDs der Vortragenden bestimmt werden können.

Ebenfalls korrekt umgesetzt wird das Kontrollbeispiel 5.4, da die Funktionale Abhängigkeit, wie gefordert, nicht erfüllt ist.

Das Vergleichsmerkmal 5 wird demnach erfüllt.

9.2.5 Wertvergleich

In diesem Ansatz wird keine Definition des Wertes eines Elements angegeben. Aus diesem Grund ist weder Vergleichsmerkmal 6 noch 7 erfüllt.

9.2.6 Geltungsbereich von Constraints

Im Ansatz von Lee, Ling und Low wird definiert, dass mit dem Header Q der Geltungsbereich eines Constraints eingegrenzt werden kann. Dies entspricht jedoch nicht der Definition eines *Context Path* (siehe Abschnitt 4.1.6), sondern definiert lediglich den *Selector* eines Constraints. Dies wird nun anhand der Kontrollbeispiele 8.3 und 8.4 verdeutlicht. Hierzu soll die Funktionale Abhängigkeit σ = (uni.institut.vortragende.vt, (lva, {@titel} $\rightarrow$ {@id})) definiert werden.

Auch hier wird die *lineage* wiederum nicht eingehalten. Dieses Problem wirkt sich jedoch für das Vergleichsmerkmal 8 stärker aus, weil, selbst wenn das `@id`-Attribut ein Kind-Knoten eines hinzugefügten Elements `vid` wäre, eine Eingrenzung des Geltungsbereichs gemäß Vergleichsmerkmal 8 nicht möglich wäre. Die Funktionale Abhängigkeit würde dabei definiert werden als σ = (//vt, [lva.@titel $\rightarrow$ vid.@id]) und zu folgendem Ergebnis führen:

lva.@titel	vid.@id
VO	L1
UE	L2
VO	L3

Diese Auflistung zeigt, dass der Constraint nicht erfüllt ist, weil die IDs der beiden Lehrveranstaltungen mit dem Titel *VO* nicht identisch sind. Dies ist im Kontrollbeispiel 8.3 nicht gefordert, sodass es nicht korrekt umgesetzt wird.

Kontrollbeispiel 8.4 würde – auf diese Weise definiert – hingegen korrekt umgesetzt werden können.

Da das Kontrollbeispiel 8.3 nicht korrekt umgesetzt werden kann, ist das Vergleichsmerkmal 8 für diesen Ansatz nicht erfüllt.

9.2.7 Nicht vorhandene Datenobjekte

Lee, Ling und Low bieten in ihrem Ansatz keine Möglichkeit, Funktionale Abhängigkeiten in XML Dokumenten mit nicht vorhandenen Datenobjekten korrekt umzusetzen. Vielmehr ignorieren sie nicht vorhandene Datenobjekte bei der Überprüfung der Constraints. Dies wird nun anhand der Kontrollbeispiele 9.4, 9.5 und 9.6 getestet. Dazu soll die Funktionale Abhängigkeit σ = (uni/vortragende/vt, [name, lva, inst $\rightarrow$ @id]) definiert werden. Hierbei wird die *lineage* wiederum nicht eingehalten, sodass eine Umsetzung in Form einer wohlstrukturierten Funktionalen Abhängigkeit nicht möglich ist. Um dennoch herauszufinden, ob der Ansatz Constraints in XML Dokumenten mit nicht vorhandenen Datenobjekten semantisch korrekt umsetzen kann, wird die fehlende *lineage* für Testzwecken nicht berücksichtigt.

Obwohl nicht vorhandene Datenobjekte in diesem Ansatz ignoriert werden, hat dies für das Kontrollbeispiel 9.4 weder unter der Annahme von *dne* noch *unk* einen negativen Einfluss auf das Ergebnis. Da die Wertemenge {Laura Leitner, UE DKE, -} nicht in den Vergleich mit einbezogen wird, kann die ID der ersten Vortragenden eindeutig bestimmt werden. Das entspricht für *dne* und *unk* dem im Kontrollbeispiel 9.4 geforderten Ergebnis.

Ebenfalls erfüllt wird die Funktionale Abhängigkeit im XML Baum in Abbildung 4.26. Dies ist zwar unter der Annahme von *dne* gewünscht, als *unk* interpretiert jedoch nicht, sodass das Kontrollbeispiel 9.5 für *unk*falsch umgesetzt wird.

Wird der Constraint nun für den XML Baum in Abbildung 4.27 verwendet, ist dieser ebenfalls erfüllt, weil beide vorhandenen Teilbäume `vt` wegen fehlender `inst`-Werte nicht beachtet werden. Dies entspricht jedoch nicht dem für das Kontrollbeispiel 9.6 vorgesehenen Ergebnis.

Aus diesem Grund sind die Vergleichsmerkmale 9.1 und 9.2 nicht erfüllt.

9.2.8 Mehrfach vorhandene Datenobjekte

Lee, Ling und Low erlauben zwar, dass Elementknoten mit demselben Label mehrfach in einem XML Dokument vorkommen dürfen, bieten jedoch keine Möglichkeit, eine semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten zu garantieren.

Dies wird nun anhand der Kontrollbeispiele 10.3 und 10.4 getestet. Dafür soll die Funktio-

nale Abhängigkeit σ = (uni.lvas.lva, {titel, vt.name, vt.sem} $\rightarrow$ {@id}) verwendet werden, welche gemäß der Definition von Lee, Ling und Low folgendermaßen lautet: σ = (/uni/lvas/lva, [titel, vt.name, vt.sem $\rightarrow$ @id]). Die Elemente bzw. Entitätstypen {uni, lvas, lva, titel, vt} bilden hier jedoch keine *lineage*, sodass die Funktionale Abhängigkeit mit diesem Ansatz nicht definiert werden kann.

Selbst wenn von nicht wohlstrukturierten Funktionalen Abhängigkeiten ausgegangen wird, d.h. die *lineage* nicht eingehalten werden muss, wäre eine semantisch korrekte Umsetzung nicht garantiert, da auch falsche Wertekombinationen möglich sind. Dies kann beim XML Baum in Abbildung 4.29 für die Elemente titel, name und sem zu den Wertemengen *{DKE, Laura Leitner, WS11}*, *{DKE, Felix Schnell, WS12}* und *{DKE, Laura Leitner, WS11}* führen. Die @id-Werte für die erste und die dritte Wertemenge sind jedoch nicht identisch, sodass die Funktionale Abhängigkeit nicht erfüllt ist. Dies entspricht für das Kontrollbeispiel 10.3 nicht dem gewünschten Ergebnis.

Selbiges gilt für das Kontrollbeispiel 10.4, da eine falsche Wertekombination zu den Wertemengen *{DKE, Laura Leitner, WS11}*, *{DKE, Felix Schnell, WS12}* und *{DKE, Felix Schnell, WS11}* führen würde. Hierbei wäre eine eindeutige Bestimmung der @id-Werte möglich. Dies ist jedoch nicht gefordert.

Da die beiden Kontrollbeispiele selbst als nicht wohlstrukturierte Funktionale Abhängigkeiten nicht korrekt umgesetzt werden können, wird das Vergleichsmerkmal 10 nicht erfüllt.

9.3 Zusammenfassung

Der Ansatz von Lee, Ling und Low ist an die Schemasprache DTD gebunden, sodass das erste Vergleichsmerkmal nicht erfüllt ist. Ebenfalls nicht erfüllt ist das Vergleichsmerkmal 2, da der Pfad im Header Q immer vom Wurzelknoten beginnen muss. Erfüllt sind hingegen die Vergleichsmerkmale 3 (Dokumentordnung) und 4 (Gemischter Inhalt), da der Ansatz auf der XML Spezifikation [17] basiert, welche die Dokumentordnung einhält und gemischten Inhalt erlaubt. Ebenfalls erfüllt ist das Vergleichsmerkmal 5, da durch die Definition Funktionaler Abhängigkeiten als $\sigma = (Q, [P_{x_1}, \cdots, P_{x_n} \rightarrow P_y])$ n-äre Constraints definiert werden können.

Nicht erfüllt sind hingegen die Vergleichsmerkmale 6 (Wertvergleich durch Verkettung von Textwerten) und 7 (Baumbasierter Wertvergleich), da keine Definition der Wertgleich-

Vergleichsmerkmale		
1	Schema	nicht erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	erfüllt
4	Baummodell: Gemischter Inhalt	erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	nicht erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	nicht erfüllt

Abbildung 9.2: Vergleichsmerkmale für den Ansatz von Lee, Ling und Low

heit zweier Elementknoten angegeben wird. Selbiges gilt für das Vergleichsmerkmal 8, da der Geltungsbereich nicht eingegrenzt werden kann.

Ebenfalls nicht erfüllt sind die letzten beiden Vergleichsmerkmale, da keine Möglichkeit geschaffen wird, Constraints in XML Dokumenten mit nicht oder mehrfach vorhandenen Datenobjekten semantisch korrekt umzusetzen.

In Abbildung 9.2 wird das Ergebnis nochmals zusammengefasst.

Kapitel 10

Ansatz von Vincent, Liu und Liu

Ein weiterer Ansatz für XML Constraints wurde von Vincent, Liu und Liu [83, 84] vorgestellt, welcher sich mit der Definition von Funktionalen Abhängigkeiten in XML beschäftigt. Erweitert wurde dieser Ansatz von Vincent, Liu und Mohania [85] (siehe Vergleichsmerkmal 10).

10.1 Kurzbeschreibung

Im Ansatz von Vincent, Liu und Liu werden Funktionale Abhängigkeiten definiert als $\sigma = p \rightarrow q$, wobei p und q jeweils für eine Menge von Pfaden steht. Im Besonderen wird auch auf die Umsetzung Funktionaler Abhängigkeiten in XML Dokumenten mit unvollständigem Inhalt eingegangen. Vincent, Liu und Liu unterscheiden in ihrem Ansatz *Strong Constraints* und *Weak Constraints*, wobei sie in ihrer Definition von *Strong Constraints* ausgehen.

Des Weiteren wird eine Vorgehensweise vorgestellt, wie Relationen als XML Bäume dargestellt werden können. Dazu wird zuerst die Relation "verschachtelt" und anschließend umgewandelt in einen XML Baum. Zusätzlich definieren Vincent, Liu und Liu eine Normalform für XML Dokumente.

10.2 Überprüfung der Vergleichsmerkmale

Im folgenden Kapitel wird die Erfüllung der einzelnen Vergleichsmerkmale durch diesen Ansatz kontrolliert.

10.2.1 Unabhängigkeit von einer Schemasprache

Vincent, Liu und Liu präsentieren einen Ansatz, Funktionale Abhängigkeiten ohne Bindung an eine DTD oder eine andere Schemasprache zu definieren. Das erste Vergleichsmerkmal ist somit erfüllt.

10.2.2 Pfadsprache

Ein Pfad wird in diesem Ansatz definiert als $p = l_1.\cdots.l_n$, mit $n \geq 1$. Herbei steht l_i, $1 \leq i \leq n$ für einen Element- oder Attributnamen oder das Textsymbol `S` und l_1 entspricht dem Wurzelknoten. Es wird also ermöglicht, dass ein Pfad mit einem Elementknoten endet. Nicht unterstützt wird jedoch die Verwendung von Kleene Operatoren.

Um die Erfüllung des Vergleichsmerkmals 2 zu prüfen, soll die Funktionale Abhängigkeit σ = `(_*.vt, {name}` $\rightarrow$ `{@id})` gemäß Kontrollbeispiele 2.3 und 2.4 definiert werden. Da der Ansatz keine Möglichkeit bietet, Kleene Operatoren bei der Definition der Constraints zu verwenden, können weder das Kontrollbeispiel 2.3 noch 2.4 umgesetzt werden. Das zweite Vergleichsmerkmal ist daher nicht erfüllt.

10.2.3 Baummodell

Vincent, Liu und Liu definieren einen Baum folgendermaßen:

Definition 10.1: *"Assume a countably infinite set **E** of element labels (tags), a countable infinite set **A** of attribute names, and a symbol `S` indicating text. An XML tree is defined to be T = (V, lab, ele, att, val, v_r) where*

- *V is a finite set of nodes in T;*
- *lab is a function from V to **E** ∪ **A** ∪ {`S`};*
- *ele is a partial function from V to a sequence of V nodes such that for any v ∈ V, if ele(v) is defined then lab(v) ∈ **E**;*
- *att is a partial function from V × **A** to V such that for any v ∈ V and l ∈ **A**, if att(v, l) = v_1 then lab(v) ∈ **E** and lab(v_1) = l;*
- *val is a function such that for any node in v ∈ V, val(v) = v if lab(v) ∈ **E** and val(v) is a string if either lab(v) = `S` or lab(v) ∈ **A**;*
- *v_r is a distinguished node in V called the root of T and we define lab(v_r) = root.*

*Since node identifiers are unique, a consequence of the definition of val is that if lab(v_1) $\in$ **E** and lab(v_2) $\in$ **E** and $v_1 \neq v_2$ then val(v_1) $\neq$ val(v_2). We also extend the definition of val to sets of nodes and if $V_1 \subseteq V$, then val(V_1) is the set defined by val(V_1) = {val(v)|v $\in$ V_1}.*

*For any v $\in$ V, if ele(v) is defined then the nodes in ele(v) are called subelements of v. For any l $\in$ **A**, if att(v, l) = v_1 then v_1 is called an attribute of v. Note that an XML tree T must be a tree. Since T is a tree, the ancestors of a node v, are denoted by Ancestor(v) and the the parent of a node v is denoted by Parent(v)."*

Laut dieser Definition besteht ein XML Baum aus einer endlichen Menge von Knoten, welche entweder ein Elementlabel oder einen Attributnamen haben, oder aber das Symbol `S` zugewiesen bekommen. Die Funktion *ele(v)* liefert eine geordnete Folge von Knoten des Elements *v* und *att(v)* liefert die Attribute eines Elementknotens. *val(v)* liefert die Knotenidentität wenn *v* ein Elementknoten ist und den Stringwert wenn *v* ein Attribut- oder Textknoten ist. v_r ist der Wurzelknoten.

Da die Funktion *ele(v)* eine geordnete Folge von Knoten des Elements *v* zurück gibt, wird die Dokumentordnung eingehalten, sodass der im Kontrollbeispiel 3.1 geforderte XML Baum abgebildet werden kann. Das Vergleichsmerkmal 3 ist daher erfüllt.

Da *ele(v)* nicht nur Elementknoten zurückgibt, sondern eine geordnete Folge beliebiger Knoten (also auch Textknoten), wird gemischter Inhalt unterstützt. Der XML Baum in Abbildung 4.8 kann somit abgebildet werden und das Vergleichsmerkmal 4 ist demnach erfüllt.

10.2.4 Arität

Wie bereits erwähnt, wird in diesem Ansatz eine Funktionale Abhängigkeit definiert als $\sigma = p \rightarrow q$, wobei p bzw. q jeweils für eine Menge von Pfaden steht. Daraus ergibt sich die erweiterte Definition $\sigma = \{p_1, ..., p_n\} \rightarrow q$, sodass n-äre Constraints ermöglicht werden.

Dies wird anhand der Kontrollbeispiele 5.3 und 5.4 getestet. Dafür soll die Funktionale Abhängigkeit σ = `(uni.vortragende.vt, {@vn, @nn}` $\rightarrow$ `{@id})` definiert werden. Dies wird durch diesen Ansatz ermöglicht, da die Menge `{@vn, @nn}` der Menge $\{p_1, ..., p_n\}$ entspricht.

Wird der Constraint für den XML Baum in Abbildung 4.9 verwendet, ist dieser wie im

Kontrollbeispiel 5.3 vorhergesehen erfüllt. Ebenfalls korrekt umgesetzt wird das Kontrollbeispiel 5.4, sodass das Vergleichsmerkmal 5 erfüllt ist.

10.2.5 Wertvergleich

Wie bereits im Abschnitt 10.2.3 beschrieben wurde, liefert die Funktion $val(v)$ angewendet auf einen Elementknoten seinen *Node Identifier*. Der Wert eines Knotens ist somit der Knoten selbst bzw. dessen Identität.

Wird diese Form des Knotenvergleichs für die Kontrollbeispiele 6.3 und 6.4 und der Funktionalen Abhängigkeit σ = `(uni.vortragende.vt, {name}` $\rightarrow$ `{@id})` verwendet, stellt sich heraus, dass das Kontrollbeispiel 6.3 auch ohne dem Wertvergleich basierend auf der Verkettung von Textwerten korrekt umgesetzt werden kann. Der Grund dafür ist, dass die Identität der beiden `name`-Elemente verschieden ist und die IDs der Vortragenden eindeutig bestimmt werden können. Für das Kontrollbeispiel 6.4 würde hingegen ein falsches Ergebnis geliefert werden, da die Funktionale Abhängigkeit aufgrund verschiedener Identität der beiden `name`-Elemente wiederum erfüllt wäre, dies aber im Kontrollbeispiel nicht vorgesehen ist. Das Vergleichsmerkmal 6 ist somit nicht erfüllt.

Der Elementknotenvergleich basierend auf der Identität der Knoten ermöglicht auch keinen baumbasierten Wertvergleich, da die Struktur und die Werte des Unterbaums beim Vergleich nicht berücksichtigt werden. Zwar ergibt sich für das Kontrollbeispiel 7.3 und der zugehörigen Funktionalen Abhängigkeit σ = `(uni.vortragende.vt, {personendaten}` $\rightarrow$ `{@id})` dennoch das richtige Ergebnis, da die beiden Elemente `personendaten` verschiedene Identitäten haben und somit zu unterscheiden sind. Für das Kontrollbeispiel 7.4 gilt dies jedoch nicht, denn auch hier wäre der Constraint erfüllt und dies entspricht nicht dem gewünschtem Ergebnis. Das Vergleichsmerkmal 7 ist somit ebenfalls nicht erfüllt.

10.2.6 Geltungsbereich von Constraints

Eine Funktionale Abhängigkeit wird in diesem Ansatz definiert als $\sigma = p \rightarrow q$, wobei p und q jeweils für einen Pfad bzw. einer Menge von Pfaden stehen. Des Weiteren werden Pfade definiert als $p = l_1. \cdots .l_n$, mit $n \geq 1$, wobei l_i für einen Elementnamen, Attributnamen oder dem Textsymbol steht. l_1 entspricht dabei dem Wurzelknoten, sodass nur Pfade definiert werden können, die direkt mit dem Wurzelknoten beginnen. Es wird daher keine Möglichkeit geboten, relative Constraints zu definieren. Die Funktionale Abhängig-

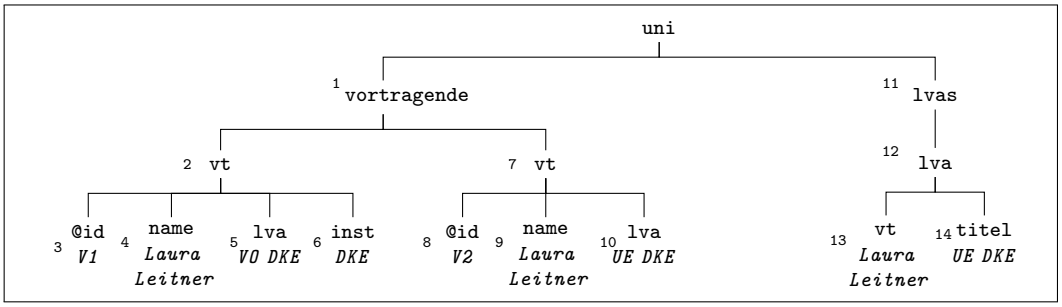

Abbildung 10.1: XML Baum 1 mit beschrifteten Knoten zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte

keit für die Kontrollbeispiele 8.3 und 8.4 kann daher nicht definiert werden, sodass das Vergleichsmerkmal 8 nicht erfüllt ist.

10.2.7 Nicht vorhandene Datenobjekte

Vincent, Liu und Liu bieten in ihrem Ansatz eine Möglichkeit, Funktionale Abhängigkeiten in XML Dokumenten mit nicht vorhandenen Datenobjekten umzusetzen. Dazu werden die XML Dokumente zuerst "erweitert", indem alle fehlenden Werte mit einem Null-Symbol $\perp$ gekennzeichnet werden. Dadurch ergibt sich eine Menge **N** = $\{\perp_1, \perp_2, \perp_3, ...\}$ aller Null-Symbole. Der Wert eines Null-Symbols ist das Null-Symbol selbst, d.h *val(v)* = $\perp$ für alle v $\in$ **N**. Somit ist der Wert eines Null-Symbols immer von den Werten aller anderen Null-Symbole zu unterscheiden. Anschließend wird der erweiterte XML Baum "vervollständigt", indem alle Null-Werte mit Datenwerten ersetzt werden.

Bezüglich der Verwendung von Constraints in unvollständigen XML Dokumenten gehen Vincent, Liu und Liu von *Strong Satisfaction* aus, d.h. jeder vervollständigte XML Baum muss den Constraint erfüllen, damit der Constraint erfüllt ist.

Der Ansatz soll nun anhand der Kontrollbeispiele 9.4, 9.5 und 9.6 unter der Annahme von *dne* und *unk* getestet werden. Dabei ist zu berücksichtigen, dass Vincent, Liu und Liu bei nicht vorhandenen Datenobjekten von *unk* ausgehen, d.h. die Daten existieren zwar in der Wirklichkeit, sind momentan jedoch nicht verfügbar.

Zur Überprüfung der Kontrollbeispiele werden zur besseren Darstellung die XML Bäume in den Abbildungen 10.1, 10.2 und 10.3 verwendet. Diese entsprechen den für die Kontrollbeispiele verwendeten XML Bäumen in den Abbildungen 4.25, 4.26 und 4.27, jedoch mit zusätzlicher Nummerierung der Knoten.

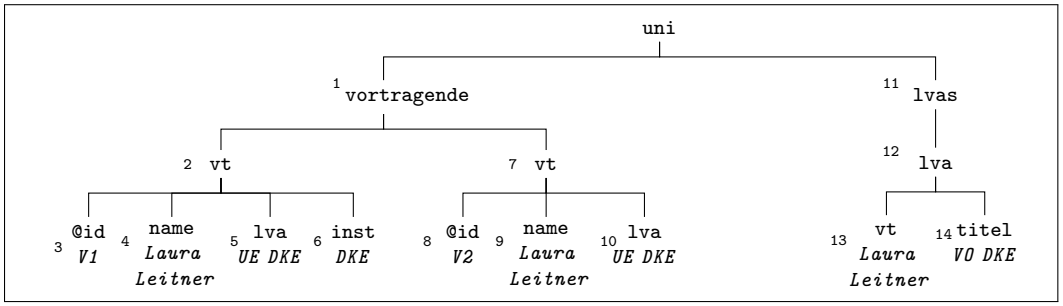

Abbildung 10.2: XML Baum 2 mit beschrifteten Knoten zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte

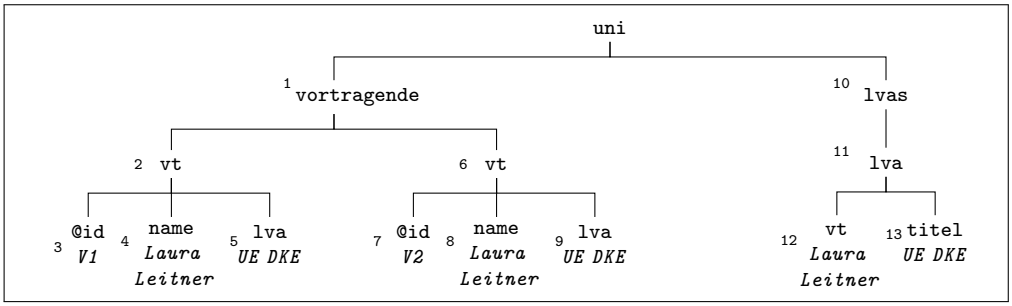

Abbildung 10.3: XML Baum 3 mit beschrifteten Knoten zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte

Für das Kontrollbeispiel 9.4 soll die Funktionale Abhängigkeit σ = (`uni.vortragende.vt`, {`name.S`, `lva.S`, `inst.S`} $\rightarrow$ {`@id`}) definiert werden. Für den XML Baum in Abbildung 10.1 führt dies zu folgendem Ergebnis:

`uni.vortragende.vt`	`name`	`lva`	`inst`	`@id`
v_2	v_4 Laura Leitner	v_5 VO DKE	v_6 DKE	v_3 V1
v_7	v_9 Laura Leitner	v_{10} UE DKE	$\perp_1$ $\perp_1$	v_8 V2

Vincent, Liu und Liu definieren als Wert eines Null-Symbols das Null-Symbol selbst. Dadurch ergeben sich die Wertemengen {*Laura Leitner, VO DKE, DKE*} und {*Laura Leitner, UE DKE,* $\perp_1$}. Die Null-Symbole unterscheiden sich zwar untereinander, d.h. der Wert eines Null-Symbols ist immer ungleich dem Wert eines anderen Null-Symbols, nicht jedoch von dem Wert eines anderen, vorhandenen Knotens. Die Werte $\perp_1$ und *DKE* sind somit gleich. Da sich jedoch die `lva`-Werte unterscheiden, kann die ID eindeutig bestimmt werden und die Funktionale Abhängigkeit ist erfüllt. Dies entspricht sowohl für *dne* als auch *unk* dem gewünschten Ergebnis.

Für das Kontrollbeispiel 9.5 wird die Funktionale Abhängigkeit nun anhand des XML Baums in Abbildung 10.2 getestet. Dies führt zu folgenden Werten:

`uni.vortragende.vt`	`name`	`lva`	`inst`	`@id`
v_2	v_4 Laura Leitner	v_5 UE DKE	v_6 DKE	v_3 V1
v_7	v_9 Laura Leitner	v_{10} UE DKE	$\perp_1$ $\perp_1$	v_8 V2

Hier ist die Funktionale Abhängigkeit nicht erfüllt, weil die Werte *DKE* und $\perp_1$ als gleich behandelt werden und die ID somit nicht eindeutig bestimmt werden kann. Dies entspricht unter der Annahme von *dne* dem gewünschten Ergebnis, für *unk* jedoch nicht.

Die Funktionale Abhängigkeit wird nun gemäß Kontrollbeispiel 9.6 für den XML Baum in Abbildung 10.3 verwendet. Dies führt zu folgendem Ergebnis:

`uni.vortragende.vt`	`name`	`lva`	`inst`	`@id`
v_2	v_4 Laura Leitner	v_5 UE DKE	$\perp_1$ $\perp_1$	v_3 V1
v_6	v_8 Laura Leitner	v_9 UE DKE	$\perp_2$ $\perp_2$	v_7 V2

Hier können die IDs eindeutig bestimmt werden, weil die Werte zweier Null-Symbole im Ansatz als ungleich definiert werden. Dies entspricht weder für *dne* noch *unk* dem gewünschten Ergebnis.

Da weder unter der Annahme von *dne* noch *unk* alle Kontrollbeispiele korrekt umgesetzt werden können, sind die Vergleichsmerkmale 9.1 und 9.2 nicht erfüllt.

10.2.8 Mehrfach vorhandene Datenobjekte

Sind Datenobjekte mehrfach vorhanden, müssen die zusammengehörigen Knoten bzw. Datenwerte gefunden werden. Dazu wird im Ansatz von Vincent, Liu und Liu das $Closest$-Prinzip verwendet, d.h. es werden jene Werte zusammengefasst, die die $Closest$-Eigenschaft erfüllen. Diese Eigenschaft wird von Knoten erfüllt, die einen gemeinsamen *Ancestor-or-Self* Knoten haben. Der *Ancestor-or-Self* Knoten ist über den Schnittpfad jener Pfade erreichbar, die zu den einzelnen Knoten führen.

Bei der Anwendung des *Closest*-Prinzips bei Funktionalen Abhängigkeiten werden die Knotenmengen folgendermaßen gebildet: Für jeden Knoten x_i der LHS wird getestet, ob er die *Closest*-Eigenschaft zum Knoten *y* der RHS erfüllt. Es wird jedoch nicht kontrolliert, ob die einzelnen Knoten der LHS zueinander auch die *Closest*-Eigenschaft erfüllen. Dies kann dazu führen, dass semantisch inkorrekte Wertemengen gebildet werden. Die Kontrollbeispiel 10.3 und 10.4 verdeutlichen dies. Dazu soll die Funktionalen Abhängigkeit σ = (`uni.lvas.lva`, {`titel.S`, `vt.name.S`, `vt.sem.S`} $\rightarrow$ {`@id`}) verwendet werden. Um die Bildung der Tupel besser zu veranschaulichen, werden in den dafür verwendeten XML Bäume die einzelnen Knoten nummeriert. Daraus ergibt sich für das Kontrollbeispiel 10.3 der in Abbildung 10.4 gezeigte XML Baum. Die Anwendung der Funktionalen Abhängigkeit führt zu folgenden Werten:

Anc.-or-Self	`titel`		`name`		`sem`		`@id`	
v_2	v_5	DKE	v_7	Laura Leitner	v_8	WS12	v_4	V1
v_2	v_5	DKE	v_{10}	Felix Schnell	v_{11}	WS11	v_4	V1
v_2	v_5	DKE	v_7	Laura Leitner	v_{11}	WS11	v_4	V1
v_2	v_5	DKE	v_{10}	Felix Schnell	v_8	WS12	v_4	V1
v_{12}	v_{15}	DKE	v_{17}	Laura Leitner	v_{18}	WS11	v_{14}	V2

Da die *Closest*-Eigenschaft einzeln für jeden Knoten der LHS zum `@id`-Knoten der RHS geprüft wird, entstehen auch die semantisch inkorrekten Wertekombinationen {*DKE, Laura Leitner, WS11*} und {*DKE, Felix Schnell, WS12*} im Teilbaum des ersten Elements `lva`. Da sowohl die richtigen als auch die falschen Wertekombinationen berücksichtigt werden, ist die Wertemenge {*DKE, Laura Leitner, WS11*} zweimal vorhanden, sie führt jedoch zu

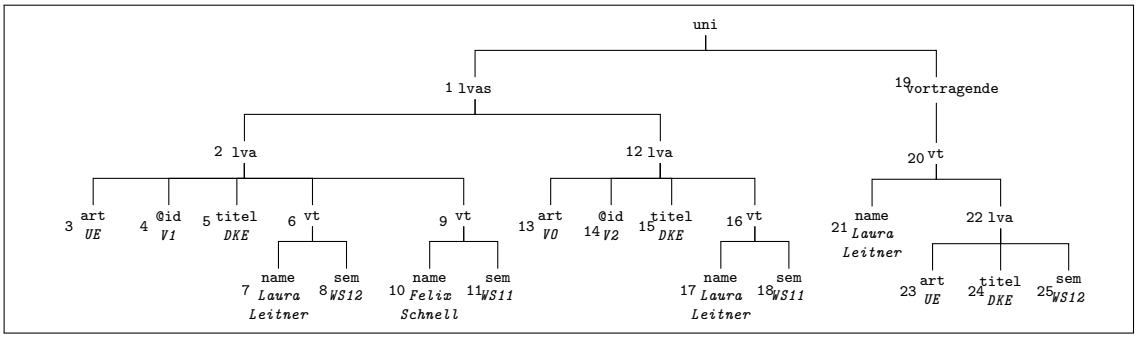

Abbildung 10.4: XML Baum 1 mit nummerierten Knoten zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten

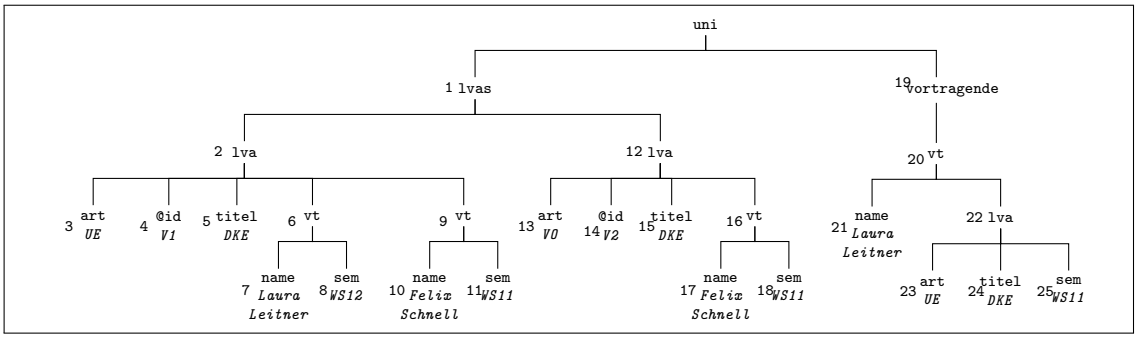

Abbildung 10.5: XML Baum 2 mit nummerierten Knoten zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten

verschiedenen ID-Werten. Die Funktionale Abhängigkeit ist daher nicht erfüllt und das Kontrollbeispiel wird falsch umgesetzt.

Hierbei ist zu beachten, dass eine semantisch korrekte Umsetzung möglich wäre, wenn es sich um unäre Constraints handelt. Dabei ist es lediglich notwendig, dass der einzige Knoten der LHS die *Closest*-Eigenschaft zum Knoten der RHS erfüllt.

Als nächstes wird diese Vorgehensweise anhand des Kontrollbeispiels 10.4 und dem XML Baum in Abbildung 10.5 getestet. Dabei ergeben sich folgende Werte:

Anc.-or-Self	`titel`		`name`		`sem`		`@id`	
v_2	v_5	DKE	v_7	Laura Leitner	v_8	WS12	v_4	V1
v_2	v_5	DKE	v_{10}	Felix Schnell	v_{11}	WS11	v_4	V1
v_2	v_5	DKE	v_7	Laura Leitner	v_{11}	WS11	v_4	V1
v_2	v_5	DKE	v_{10}	Felix Schnell	v_8	WS12	v_4	V1
v_{12}	v_{15}	DKE	v_{17}	Felix Schnell	v_{18}	WS11	v_{14}	V2

Auch hier wurden wiederum falsche Wertekombinationen gebildet. Da bei der Prüfung des Constraints sowohl die richtigen als auch die falschen Wertekombinationen berücksichtigt werden, ist die Wertemenge {*DKE, Felix Schnell, WS11*} zweimal vorhanden. Die Funktionale Abhängigkeit ist daher nicht erfüllt. Dies entspricht dem im Kontrollbeispiel 10.4 geforderten Ergebnis.

Da das Kontrollbeispiel 10.3 nicht korrekt umgesetzt werden kann, ist das Vergleichsmerkmal 10 nicht erfüllt.

Vincent, Liu und Mohania [85] erweiterten diese Definition, sodass semantisch korrekte Kombinationen garantiert werden. Dabei wird die *Closest*-Eigenschaft nicht nur einzeln für jeden Knoten der LHS mit dem Knoten der RHS geprüft, sondern auch die Knoten der LHS müssen untereinander die *Closest*-Eigenschaft erfüllen. Diese Vorgehensweise wird nun anhand der Kontrollbeispiele 10.3 und 10.4 getestet.

Für das Kontrollbeispiel 10.3 und den XML Baum in Abbildung 10.4 ergeben sich dadurch folgende Kombinationen:

Anc.-or-Self	`titel`		`name`		`sem`		`@id`	
v_2	v_5	DKE	v_7	Laura Leitner	v_8	WS12	v_4	V1
v_2	v_5	DKE	v_{10}	Felix Schnell	v_{11}	WS11	v_4	V1
v_{12}	v_{15}	DKE	v_{17}	Laura Leitner	v_{18}	WS11	v_{14}	V2

Hierbei wurden nur semantisch korrekte Kombinationen gebildet, da auch geprüft wird, ob die Knoten `titel`, `name` und `sem` untereinander die *Closest*-Eigenschaft erfüllen. Die Funktionale Abhängigkeit wird daher erfüllt und entspricht dem im Kontrollbeispiel 10.3 geforderten Ergebnis.

Für das Kontrollbeispiel 10.4 wird die Funktionale Abhängigkeit σ = `(uni.lvas.lva, {titel.S, vt.name.S, vt.sem.S}` $\rightarrow$ `{@id})` anhand des XML Baums in Abbildung 10.5 getestet. Dabei ergeben sich folgende Kombinationen:

Anc.-or-Self	`titel.S`		`name.S`		`sem.S`		`@id`	
v_2	v_5	DKE	v_7	Laura Leitner	v_8	WS12	v_4	V1
v_2	v_5	DKE	v_{10}	Felix Schnell	v_{11}	WS11	v_4	V1
v_{12}	v_{15}	DKE	v_{17}	Felix Schnell	v_{18}	WS11	v_{14}	V2

Auch hier erfüllen die Knoten `titel, name` und `sem` untereinander sowie zum Knoten `@id` die *Closest*-Eigenschaft. Die Funktionale Abhängigkeit ist dabei nicht erfüllt, da die Wertemenge {*DKE, Felix Schnell, WS11*} zweimal vorhanden ist, jedoch zu verschiedenen ID-Werten führt. Dies entspricht dem gewünschten Ergebnis, sodass das Kontrollbeispiel 10.4 korrekt umgesetzt wird und das Vergleichsmerkmal 10 ist erfüllt.

10.3 Zusammenfassung

Der Ansatz von Vincent, Liu und Liu ist unabhängig von einer Schemasprache anwendbar, sodass das erste Kriterium erfüllt ist. Das Vergleichsmerkmal 2 ist hingegen nicht erfüllt, da die Verwendung von Kleene Operatoren nicht unterstützt wird.

Das Baummodell betreffend stellt sich heraus, dass das Vergleichsmerkmal 3 erfüllt ist, da die Dokumentordnung eingehalten wird. Selbiges gilt für das Vergleichsmerkmal 4, da gemischter Inhalt in XML Dokumenten ermöglicht wird. Das Aritäts-Merkmal ist ebenfalls erfüllt, da durch die Definition einer Funktionalen Abhängigkeit als $\sigma = \{p_1, ..., p_n\} \rightarrow q$ n-äre Constraints definiert werden können.

Als Wertvergleich ermöglicht der Ansatz lediglich den Vergleich von Elementknoten auf Grundlage ihrer Identität. Der Wertvergleich von gemischtem Inhalt sowie der baumbasierten Wertvergleich werden nicht unterstützt, sodass die Vergleichsmerkmale 6 und 7 nicht erfüllt sind. Das nächste Vergleichsmerkmal, die Definition von relativen Constraints, wird ebenfalls nicht erfüllt, da keine Einschränkung des Geltungsbereichs ermöglicht wird.

Vergleichsmerkmale		
1	Schema	erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	erfüllt
4	Baummodell: Gemischter Inhalt	erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	nicht erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	nicht erfüllt [83, 84] erfüllt [85]

Abbildung 10.6: Vergleichsmerkmale für den Ansatz von Vincent, Liu und Liu

Beim Umgang mit nicht vorhandenen Datenobjekten werden falsche Ergebnisse erzielt, sodass die Vergleichsmerkmale 9.1 und 9.2 nicht erfüllt sind. Ebenfalls nicht erfüllt ist das Vergleichsmerkmal 10, da es trotz Verwendung des *Closest*-Prinzips möglich ist, dass semantisch inkorrekte Kombinationen gebildet werden. Der Ansatz wurde jedoch von Vincent, Liu und Mohania [85] erweitert, sodass eine semantisch korrekte Umsetzung ermöglicht wird und das Vergleichsmerkmal 10 somit erfüllt ist.

Abbildung 10.6 fasst die Ergebnisse nochmals zusammen.

Ergänzend zu [83, 84] bieten Vincent, Liu und Liu in [59] eine Möglichkeit, lokale Funktionale Abhängigkeiten zu definieren, sodass der Geltungsbereich eingeschränkt werden kann. Des Weiteren stellen sie in [80, 81] einen Ansatz für Mehrwertige Abhängigkeiten vor.

Kapitel 11

Ansatz von Vincent et al.

Ein Ansatz für Inklusionsabhängigkeiten in XML wurde von Vincent et al. [86] entwickelt. Dieser wird in [49] als *Intersection Path Approach* bezeichnet und im folgenden Kapitel vorgestellt.

11.1 Kurzbeschreibung

Vincent et al. definieren eine Inklusionsabhängigkeit für XML (XIND) als $\sigma = P[p_1, \cdots, p_n] \subseteq Q[q_1, \cdots, q_n]$, mit P bzw. Q als *Selector* und $[p_1, \cdots, p_n]$ bzw. $[q_1, \cdots, q_n]$ als *Fields*. Des Weiteren wird für eine Menge von Pfaden $p_1, \cdots, p_n$, $n \leq 1$, jener Pfad P definiert, der sich aus der Schnittmenge der Pfade $p_1, \cdots, p_n$ ergibt. Dieser Pfad wird anschließend als *Selector* bei der Definition der XINDs verwendet.

Darüber hinaus gehen Vincent et al. auf die Beziehung zwischen Inklusionsabhängigkeiten in Relationalen Datenbanken und Inklusionsabhängigkeiten in XML Dokumenten ein und präsentieren ein Axiomensystem[14] zur Analyse des Implikationsproblems.

11.2 Überprüfung der Vergleichsmerkmale

Im Folgenden wird der Ansatz anhand der im Kapitel 4 vorgestellten Vergleichsmerkmale analysiert.

[14] Ein Axiom wird in [46] folgendermaßen definiert: "Eine Annahme, die nicht aus logisch früheren Sätzen abgeleitet werden kann und die als Grundlage für eine wissenschaftliche Aussage dient."

11.2.1 Unabhängigkeit von einer Schemasprache

Im Ansatz von Vincent et al. wird kein Vorhandensein einer Schemasprache vorausgesetzt. Vielmehr basiert der Ansatz auf einer vorhandenen Menge von Pfaden. Daher ist das Vergleichsmerkmal 1 erfüllt.

11.2.2 Pfadsprache

Vincent et al. definieren einen Pfad als $P = l_1. \cdots .l_n$, wobei $l_i, 1 \leq i \leq n$, für einen Element- oder Attributnamen oder dem Textsymbol `S` steht und l_1 repräsentiert den Wurzelknoten.

Aus dieser Definition ist bereits ersichtlich, dass nur einfache Abwärtspfade verwendet werden können und Kleene Operatoren nicht unterstützt werden. Des Weiteren darf ein Pfad nicht mit einem Elementknoten enden, da l_n entweder ein Text- oder Attributknoten sein muss. Das 2. Vergleichsmerkmal ist daher nicht erfüllt.

11.2.3 Baummodell

Dieser Ansatz verwendet ein an Buneman et al. [19] angelehntes Baummodell. Ein Baum wird folgendermaßen definiert:

Definition 11.1: *"Assume a countably infinite set **E** of element labels (tags), a countable infinite set **A** of attribute names and a symbol `S` indicating text. An XML tree is defined to be T = (V, lab, ele, att, val, vr) where*

- *V is a finite set of nodes in T;*
- *lab is a function from V to **E** ∪ **A** ∪ {`S`};*
- *ele is a partial function from V to a sequence of V nodes such that for any v ∈ V, if ele(v) is defined then lab(v) ∈ **E**;*
- *att is a partial function from V × **A** to V such that for any v ∈ V and l ∈ **A**, if att(v, l) = v_1 then lab(v) ∈ **E** and lab(v_1) = l;*
- *val is a function such that for any node in v ∈ V, val(v) = v if lab(v) ∈ **E** and val(v) is a string if either lab(v) = S or lab(v) ∈ **A**;*
- *v_r is a distinguished node in V called the root of T and we define lab(v_r) = root.*

Since node identifiers are unique, a consequence of the definition of val is that if lab(v_1)

*$\in$ **E** and lab(v_2) $\in$ **E** and $v_1 \neq v_2$ then val(v_1) $\neq$ val(v_2). We also extend the definition of val to sets of nodes and if $V_1 \subseteq$ V, then val(V_1) is the set defined by val(V_1) = {val(v)|v $\in$ V_1}.*

*For any v $\in$ V, if ele(v) is defined then the nodes in ele(v) are called subelements of v. For any l $\in$ **A**, if att(v, l) = v_1 then v_1 is called an attribute of v. Note that an XML tree T must be a tree. Since T is a tree the ancestors of a node v, are denoted by Ancestor(v) and the the parent of a node v is denoted by Parent(v)."*

Ein Baum besteht somit aus einer endlichen Menge von Knoten, welche entweder ein Elementlabel, einen Attributnamen oder das Textsymbol `S` haben. Mit der Funktion *ele(v)* wird eine geordnete Folge von Knoten zurückgegeben. *att(v)* liefert die Attribute des Knotens *v*. Der Wert eines Knotens wird mit der Funktion *val(v)* zurück gegeben, wobei dieser für Elementknoten der Knoten selbst und für Attribut- und Textknoten ein Stringwert ist. Der Wurzelknoten wird mit v_r definiert.

Da mit der Funktion *ele(v)* eine geordnete Folge von Unterknoten zurück gegeben wird, wird die Dokumentordnung eingehalten. Der XML Baum in Abbildung 4.6 kann somit abgebildet werden und das Vergleichsmerkmal 3 ist erfüllt.

Des Weiteren wird gemischter Inhalt in einem XML Dokument ermöglicht, da die Funktion *ele(v)* eine geordnete Folge von Unterelementen zurück gibt, welche sowohl Element- als auch Textknoten beinhalten kann. Der XML Baum in Abbildung 4.8 kann somit abgebildet werden und das Vergleichsmerkmal 4 ist erfüllt.

11.2.4 Arität

Eine XIND wird definiert als $\sigma = P[p_1, \cdots, p_n] \subseteq Q[q_1, \cdots, q_n]$, sodass mit $[p_1, \cdots, p_n]$ und $[q_1, \cdots, q_n]$ n-äre Constraints ermöglicht werden. Die für die Kontrollbeispiele 5.5 und 5.6 vorgesehene Inklusionsabhängigkeit σ = `(uni.lvas.lva.vt, [@vn, @nn])` $\subseteq$ `(uni.vortragende.vt, [@vn, @nn])` kann somit definiert werden. Begründet ist dies auch darin, dass die beiden *Selector*-Pfade `uni.lvas.lva.vt` und `uni.vortragende.vt` die Schnittmenge der jeweiligen Attributknoten `@vn` und `@nn` in der LHS bzw. RHS bilden. Folgende Abbildung soll dies verdeutlichen:

LHS	`uni.lvas.lva.vt`
	`uni.lvas.lva.vt.@vn`
	`uni.lvas.lva.vt.@nn`
RHS	`uni.vortragende.vt`
	`uni.vortragende.vt.@vn`
	`uni.vortragende.vt.@nn`

Da eine korrekte Umsetzung von diesem Ansatz ermöglicht wird, ist das Vergleichsmerkmal 5 erfüllt.

11.2.5 Wertvergleich

Vincent et al. definieren den Wert eines Knotens mit der im Baummodel angegebenen Funktion *val(v)*. Diese Funktion liefert für einen Attribut- oder Textknoten den Stringwert dieses Knotens und gibt für Elementknoten den Knoten selbst bzw. dessen *Node Identifier* zurück, sodass *val(v) = v*. Darüber hinaus wird keine weitere Definition einer Wertgleichheit angegeben, sodass die Vergleichsmerkmale 6 und 7 nicht erfüllt sind.

11.2.6 Geltungsbereich

Vincent et al. bieten keine Möglichkeit, den Geltungsbereich eines Constraints einzugrenzen. Zwar wird mit den *Selector*-Pfaden die Schnittmenge der *Field*-Pfade abgebildet, dies kann jedoch nicht zur Begrenzung des Geltungsbereich verwendet werden.

Der für die Kontrollbeispiele 8.7 und 8.6 vorgesehen Constraint σ = `(uni.institut, (vortragende.vt.lva, [@titel])` $\subseteq$ `(lvas.lva, [@titel]))` kann daher nicht definiert werden und das Vergleichsmerkmal 8 ist nicht erfüllt.

11.2.7 Nicht vorhandene Datenobjekte

Im Ansatz von Vincent et al. werden Tupel mit nicht vorhandenen Datenobjekten bei der Prüfung von Constraints ignoriert. Diese Vorgehensweise wird nun anhand der Kontrollbeispiele 9.7 und 9.8 getestet. Dazu soll die Inklusionsabhängigkeit σ = `(uni.lvas.lva, [vt.S, titel.S, inst.S])` $\subseteq$ `(vortragende.vt, [name.S, lva.S, inst.S])` definiert werden.

Wird die Inklusionsabhängigkeit anhand dem XML Baum in Abbildung 4.25 getestet, ist sie erfüllt. Der Grund dafür ist, dass der `inst`-Knoten auf der LHS fehlt und die sich

daraus ergebende Liste [*Laura Leitner, UE DKE,* -] ignoriert wird. Es ist somit keine Teilmengenbeziehung zu prüfen, da im Teilbaum `lvas` keine vollständige Liste vorhanden ist. Dieses Ergebnis ist im Kontrollbeispiel 9.7 zwar für *dne* vorgesehen, für *unk* jedoch nicht.

Ebenfalls aus gleichem Grund erfüllt ist der Constraint für den XML Baum in Abbildung 4.26. Da dies jedoch weder unter *dne* noch *unk* gefordert ist, wird das Kontrollbeispiel 9.8 nicht semantisch korrekt umgesetzt. Die Vergleichsmerkmale 9.1 und 9.2 sind somit nicht erfüllt.

11.2.8 Mehrfach vorhandene Datenobjekte

Mehrfach vorhandene Datenobjekte in XML Dokumenten werden zwar in Vincent et al. nicht verboten, es wird jedoch keine Möglichkeit angegeben, semantisch korrekte Verknüpfungen zu garantieren.

Angewendet auf das Kontrollbeispiel 10.5 und die XIND σ = `(uni.vortragende.vt [name.S, lva.titel.S, lva.sem.S])` $\subseteq$ `(uni.lvas.lva, [vt.name.S, titel.S, vt.sem.S])` würde das zu folgenden Wertekombinationen führen:

Teilbaum `vortragende` | Teilbaum `lvas`

`name.S`	`lva.titel.S`	`lva.sem.S`	`vt.name.S`	`titel.S`	`vt.sem.S`
Laura Leitner	DKE	WS12	Laura Leitner	DKE	WS12
–	–	–	Laura Leitner	DKE	WS11
–	–	–	Felix Schnell	DKE	WS12
–	–	–	Felix Schnell	DKE	WS11
–	–	–	Laura Leitner	DKE	WS11

Auch mit semantisch inkorrekten Verknüpfungen der Werte wäre der Constraint wie gewünscht erfüllt, da die Wertemenge {*Laura Leitner, DKE, WS12*} der LHS auch auf der RHS vorhanden ist. Für das Kontrollbeispiel 10.6 ergibt sich jedoch ein falsches Ergebnis, weil sich folgende Kombinationen ergeben:

Teilbaum `vortragende`			Teilbaum `lvas`		
`name.S`	`lva.titel.S`	`lva.sem.S`	`vt.name.S`	`titel.S`	`vt.sem.S`
Laura Leitner	DKE	WS11	Laura Leitner	DKE	WS12
–	–	–	Laura Leitner	DKE	WS11
–	–	–	Felix Schnell	DKE	WS12
–	–	–	Felix Schnell	DKE	WS11
–	–	–	Felix Schnell	DKE	WS11

Hier wäre der Constraint erfüllt. Dies entspricht jedoch nicht dem im Kontrollbeispiel gefordertem Ergebnis.

Das Vergleichsmerkmal 10 ist somit nicht erfüllt.

11.3 Zusammenfassung

Abbildung 11.1 fasst das Ergebnis nochmals zusammen.

Der Ansatz ist an keine Schemasprache gebunden, sodass das Vergleichsmerkmal 1 erfüllt ist. Nicht erfüllt ist hingegen das Vergleichsmerkmal 2, da nur einfache Abwärtspfade verwendet werden können, welche mit einem Attribut- oder Textknoten enden müssen.

Bezüglich des Baummodells stellt sich heraus, dass die Dokumentordnung eingehalten und gemischter Inhalt ermöglicht wird. Die Vergleichsmerkmale 3 und 4 sind daher erfüllt. Selbiges gilt für das Vergleichsmerkmal 5, da auf Grund der Definition einer XIND als $\sigma = P[p_1, \cdots, p_n] \subseteq Q[q_1, \cdots, q_n]$ n-äre Constraints ermöglicht werden.

Nicht erfüllt sind hingegen die Vergleichsmerkmale 6 und 7, da der Wert eines Elementknotens der Knoten selbst (bzw. dessen *Node Identifier*) ist. Dadurch wird keine Möglichkeit geboten, Elemente auf Grundlage der Verkettung von Textwerten oder baumbasiert zu vergleichen. Ebenfalls nicht erfüllt ist das Vergleichsmerkmal 8, da der Geltungsbereich nicht eingeschränkt werden kann.

Sind in einem XML Dokument Datenobjekte nicht vorhanden, kann in diesem Ansatz eine semantisch korrekte Umsetzung nicht garantiert werden, sodass die Vergleichsmerkmale 9.1 und 9.2 nicht erfüllt sind. Selbiges gilt für das Vergleichsmerkmal 10, da keine Möglichkeit geboten wird, semantisch korrekte Kombinationen der Werte zu garantieren.

Vergleichsmerkmale		
1	Schema	erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	erfüllt
4	Baummodell: Gemischter Inhalt	erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	nicht erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	nicht erfüllt

Abbildung 11.1: Vergleichsmerkmale für den Ansatz von Vincent et al.

Kapitel 12

Ansatz von Hartmann, Link und Kirchberg

Ein weiterer Ansatz für Funktionale Abhängigkeiten in XML wurde von Hartmann und Link [39] bzw. Hartmann, Link und Kirchberg [43] entwickelt. Dieser wird im folgendem Kapitel analysiert.

12.1 Kurzbeschreibung

Hartmann, Link und Kirchberg definieren einen Ansatz für Funktionale Abhängigkeiten in XML mittels Sub-Graphen anstatt mit Pfaden. Ein XML Baum wird dabei als sogenannter *Rooted Graph* definiert, d.h. ein Graph der einen Wurzelknoten hat und dessen Kanten bzw. Knoten alle genau mit einem Pfad vom Wurzelknoten aus erreicht werden können. Des Weiteren wird ein XML Schema Graph[15] verwendet um die Struktur eines Baumes zu definieren.

Eine Funktionale Abhängigkeit wird in diesem Ansatz definiert als $\sigma = v : X \rightarrow Y$ wobei *v* einen Knoten im XML Schema Baum *T* darstellt und *X* und *Y* jeweils einen Subgraphen von *v*.

[15] Ein Schema Graph ist nicht zu verwechseln mit der Schemasprache XML Schema [32].

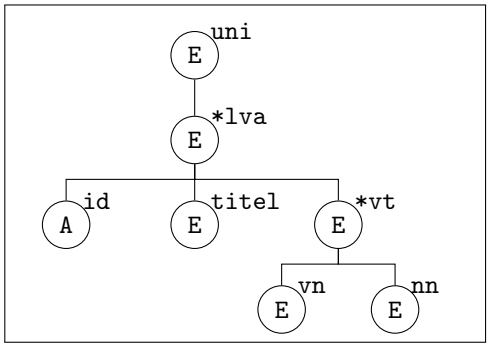

Abbildung 12.1: XML Schema Baum für den XML Baum in Abbildung 12.2

12.2 Überprüfung der Vergleichsmerkmale

Mit Hilfe der Vergleichsmerkmale in Kapitel 4 wird der Ansatz von Hartmann, Link und Kirchberg nun analysiert.

12.2.1 Unabhängigkeit von einer Schemasprache

Wie bereits erwähnt, verwenden Hartmann, Link und Kirchberg einen XML Schema Baum zur Definition der Baumstruktur. Dort werden die Knoten, welche im XML Baum vorkommen sollen, sowie deren Häufigkeit definiert. Ein Knoten kann dabei folgende Häufigkeiten haben:

1 Knoten kommt genau einmal vor
? Knoten kommt höchstens einmal vor
* Knoten kommt null oder beliebig oft vor

Dies wird anhand des nachfolgenden Beispiels verdeutlicht:

Beispiel 12.1: *Der XML Baum in Abbildung 12.2 zeigt die Daten zweier Lehrveranstaltungen, mit den Unterknoten* `@id, titel` *und einer unbestimmten Menge von* `vt`*-Knoten. Abbildung 12.1 zeigt den zugehörigen XML Schema Baum. Hier werden die Häufigkeiten der Knoten, d.h. 1 oder *, angegeben. Die Bezeichnungen E bzw. A weisen auf die Knotenart hin, d.h. Element- und Attributknoten.*

Da dieser Ansatz vom Vorhandensein eines Schemabaumes ausgeht, ist das Vergleichsmerkmal 1 nicht erfüllt.

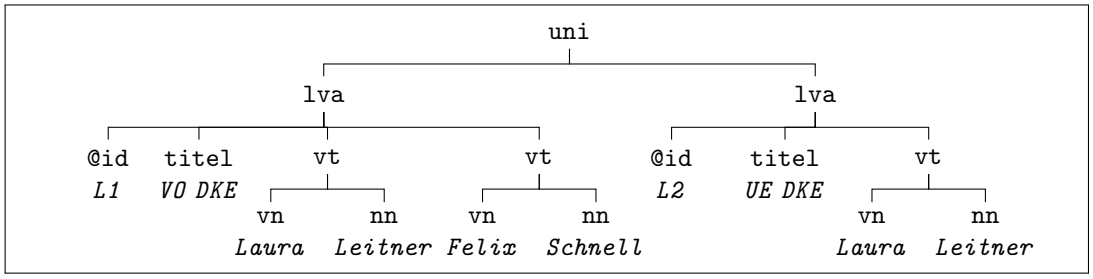

Abbildung 12.2: XML Baum, entsprechend dem XML Schema Baum in Abbildung 12.1

12.2.2 Pfadsprache

Der hier vorgestellte Subgraph-Ansatz greift auf die Knoten nicht mit Pfadsprachen zu, sondern definiert die Subgraphen *X* und *Y* in der Funktionalen Abhängigkeit direkt. Aus diesem Grund ist das Vergleichsmerkmal 2 nicht erfüllt.

12.2.3 Baummodell

Ein XML Baum wird in diesem Ansatz als gerichteter Graph $T = (V_T, A_T)$ definiert, wobei V_T für die Kanten bzw. Knoten und A_T für die Achsen bzw. Pfade steht.

Die Nachfolger eines Knotens werden in diesem Ansatz als ungeordnet behandelt, sodass das Vergleichsmerkmal 3 nicht erfüllt ist.

Des Weiteren wird mit der Funktion *freq* die Häufigkeit definiert, wie oft ein Knoten vorhanden sein darf. Für Elementknoten gilt hier, dass sie mehrere Unterelemente haben können oder genau einen. Textwerte werden hier nicht als Knoten behandelt, sondern als Wert eines Elementknotens, der sich an der letzten Stelle eines Pfades befindet, d.h. ein Blattknoten ist. Gemischter Inhalt wird somit nicht unterstützt und Vergleichsmerkmal 4 ist daher ebenfalls nicht erfüllt.

12.2.4 Arität

Durch eine Erweiterung ihres Ansatzes ermöglichen Hartmann, Link und Kirchberg die Definition von n-ären Constraints. Um deren Umsetzung zu analysieren, werden die Kontrollbeispiele 5.3 und 5.4 verwendet. Dafür soll die Funktionale Abhängigkeit σ = (uni.vortragende.vt, {@vn, @nn} $\rightarrow$ {@id}) definiert werden. Dies wird umgesetzt mit $v_{vt} : X \rightarrow Y$, wobei *X* der v_{vt}-Subgraph mit den Blattknoten $v_{@vn}$ und $v_{@nn}$ ist und *Y* der

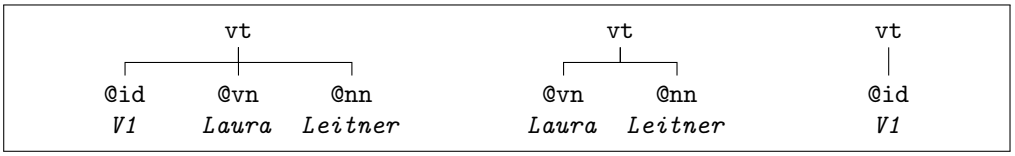

Abbildung 12.3: v_{vt}-Subgraphen für das Kontrollbeispiel 5.3 - Teil 1

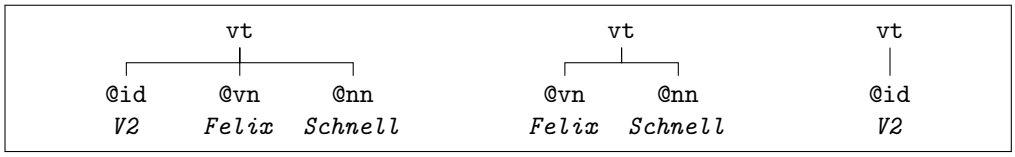

Abbildung 12.4: v_{vt}-Subgraphen für das Kontrollbeispiel 5.3 - Teil 2

v_{vt}-Subgraph mit dem Blattknoten $v_{@id}$.

Für das Kontrollbeispiel 5.3 ergeben sich dadurch die in den Abbildungen 12.3 und 12.4 angegebenen v_{vt}-Bäume sowie die v_{vt}-Subgraphen mit den Blattknoten $v_{@vn}$ und $v_{@nn}$ bzw. $v_{@id}$. Die Funktionale Abhängigkeit ist erfüllt, weil die v_{vt}-Subgraphen mit den Blattknoten $v_{@vn}$ und $v_{@nn}$ die v_{vt}-Subgraphen mit den Blattknoten $v_{@id}$ eindeutig bestimmen können. Dies entspricht dem geforderten Ergebnis.

Wird diese Vorgehensweise nun anhand dem Kontrollbeispiel 5.4 getestet, führt dies zu den in den Abbildungen 12.5 und 12.6 angegebenen v_{vt}-Bäumen sowie den zugehörigen v_{vt}-Subgraphen mit den Blattknoten $v_{@vn}$ und $v_{@nn}$ bzw. $v_{@id}$.

Hierbei haben die beiden v_{vt}-Subgraphen mit den Blattknoten $v_{@vn}$ und $v_{@nn}$ dieselben Werte, sodass die IDs der Vortragenden nicht eindeutig bestimmt werden können. Dies entspricht ebenfalls dem gewünschten Ergebnis. Das Vergleichsmerkmal 5 ist demnach erfüllt.

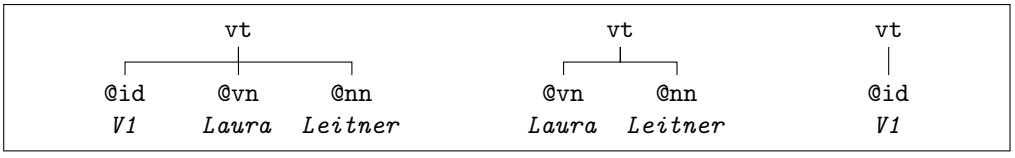

Abbildung 12.5: v_{vt}-Subgraphen für das Kontrollbeispiel 5.2 - Teil 1

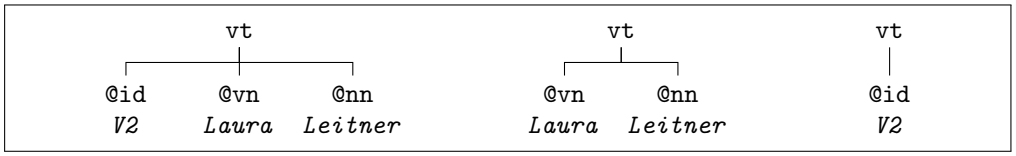

Abbildung 12.6: v_{vt}-Subgraphen für das Kontrollbeispiel 5.4 - Teil 2

12.2.5 Wertvergleich

Da von diesem Ansatz das Vorhandensein von gemischtem Inhalt in einem XML Dokument nicht unterstützt wird (siehe Vergleichsmerkmal 4), wird auch der Wertvergleich basierend auf der Verkettung von Textwerten nicht ermöglicht. Das Vergleichsmerkmal 6 ist daher nicht erfüllt.

Im Gegensatz dazu ermöglichen Hartmann, Link und Kirchberg die Verwendung von Wertgleichheit beim Vergleich zweier Elementknoten bzw. ihrer Subgraphen. Zwei Subgraphen sind dabei gleich, wenn sie dieselbe Struktur haben und die Werte ihrer Blattknoten (ermittelt mit der Funktion *val*) gleich sind.

Wird dies angewendet auf die Kontrollbeispiele 7.3 und 7.4 ergibt sich die Funktionale Abhängigkeit $v_{vt} : X \rightarrow Y$, mit dem v_{vt}-Subgraphen $v_{personendaten}$ als *X* und den v_{vt}-Subgraphen $v_{@id}$ als *Y*. Im XML Baum in Abbildung 4.17 unterscheiden sich sowohl die Werte als auch die Struktur der Subgraphen $v_{personendaten}$, sodass die IDs eindeutig bestimmt werden können. Beim XML Baum in Abbildung 4.18 ist dies jedoch nicht der Fall und der Constraint ist, wie vorgesehen, nicht erfüllt.

Da beide Kontrollbeispiele zum richtigen Ergebnis führen, ist das Vergleichsmerkmal 7 erfüllt.

12.2.6 Geltungsbereich

Mit der Definition einer Funktionale Abhängigkeit als $v : X \rightarrow Y$ wird mit *v* jener Knoten definiert, dessen Subgraphen *X* und *Y* die Funktionale Abhängigkeit erfüllen sollen. Dadurch wird aber keine adäquate Möglichkeit geboten, den Geltungsbereich der Constraints einzuschränken. Dies wird durch die Kontrollbeispiele 8.3 und 8.4 verdeutlicht.

Dazu soll die Funktionale Abhängigkeit σ = `(uni.institut.vortragende.vt, (lva, {@titel} → {@id}))` definiert werden. Gemäß Hartmann, Link und Kirchberg führt das zu $v_{lva} : X \rightarrow Y$, mit dem v_{vt}-Subgraphen v_{titel} als *X* und $v_{@id}$ als *Y*. Dies stellt

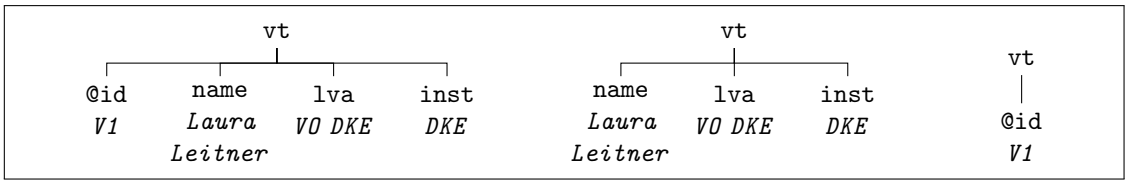

Abbildung 12.7: v_{vt}-Subgraphen für das Kontrollbeispiel 9.4

jedoch einen absoluten Constraint dar, da keine Eingrenzung auf die Lehrveranstaltungen eines bestimmten Vortragenden möglich ist.

Die Kontrollbeispiele können somit nicht korrekt umgesetzt werden und das Vergleichsmerkmal 8 ist nicht erfüllt.

12.2.7 Nicht vorhandene Datenobjekte

Durch die Definition der Häufigkeit im XML Schema Baum, wird angegeben, wie oft ein Element oder Attribut im XML Baum vorkommen muss. Dabei ist es auch möglich, dass als Häufigkeit * gewählt wird, d.h. ein Knoten kann null mal oder beliebig oft vorhanden sein. Generell wird somit erlaubt, dass Datenobjekte in XML Dokumenten nicht vorhanden sind.

Bezüglich der Funktionalen Abhängigkeiten wird jedoch eine Einschränkung vorgenommen. Für die v-Subgraphen von *X* gilt, dass sie vollständig sein müssen. Jene Subgraphen mit nicht vorhandenen Datenobjekten werden ignoriert.

Wird dies nun anhand dem Kontrollbeispiel 9.4 getestet, werden lediglich der v_{vt}-Subgraph des ersten Elements `vt` berücksichtigt. Dieser wird in Abbildung 12.7 dargestellt. Die Funktionale Abhängigkeit ist erfüllt, da die ID der Vortragenden eindeutig bestimmt werden kann. Dies entspricht sowohl für *dne* als auch *unk* dem gewünschten Ergebnis.

Beim Kontrollbeispiel 9.5 führt diese Vorgehensweise zum selben Ergebnis, da wiederum nur der v_{vt}-Subgraph des ersten Elements `vt` berücksichtigt wird. Dieser wird in Abbildung 12.8 gezeigt. Unter der Annahme von *dne* führt dies zum richtigen Ergebnis, weil die Funktionale Abhängigkeit erfüllt ist. Für *unk* ist dies jedoch nicht gefordert.

Als nächstes wird der Ansatz anhand des Kontrollbeispiels 9.6 getestet. Hier werden beide v_{vt}-Subgraphen ignoriert, weil deren `inst`-Elemente fehlen. Die Funktionale Abhängigkeit ist somit erfüllt. Dies ist weder für *dne* noch *unk* gefordert, sodass das Kontrollbei-

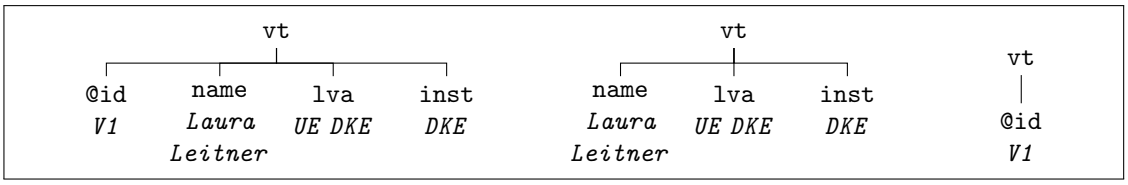

Abbildung 12.8: v_{vt}-Subgraphen für das Kontrollbeispiel 9.5

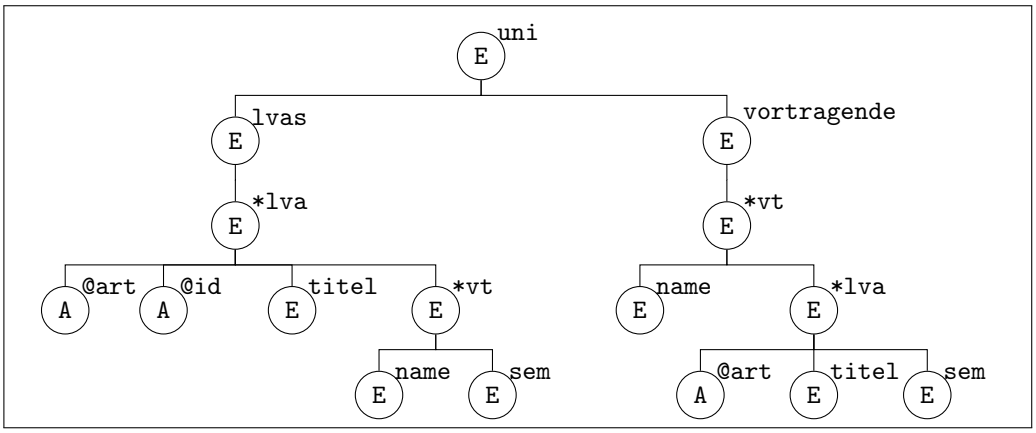

Abbildung 12.9: XML Schema Baum für die XML Bäume in den Abbildungen 4.29 und 4.30

spiel 9.6 falsch umgesetzt wird.

Die Vergleichsmerkmale 9.1 und 9.2 sind daher nicht erfüllt.

12.2.8 Mehrfach vorhandene Datenobjekte

Mit Hilfe der Funktion *freq* wird im XML Schema Baum festgelegt, wie oft ein Knoten vorhanden sein darf. Mit der Häufigkeit * wird erlaubt, dass Knoten mehrfach vorhanden sind.

Für die Kontrollbeispiele 10.3 und 10.4 soll nun die Funktionale Abhängigkeit σ = (uni.lvas.lva, {titel, vt.name, vt.sem} $\rightarrow$ {@id}) verwendet werden. Dazu muss zuerst der dazugehörige XML Schema Baum definiert werden. Dieser ist in Abbildung 12.9 angegeben.

Wird die Funktionale Abhängigkeit nun gemäß Hartmann, Link und Kirchberg umgesetzt werden, lautet sie: $v_{lva} : X \rightarrow Y$, wobei X den v_{lva}-Subgraphen mit den Blattknoten v_{titel}, v_{name} und v_{sem} bezeichnet und Y ist der Subgraph $v_{@id}$. Daraus ergeben sich für das

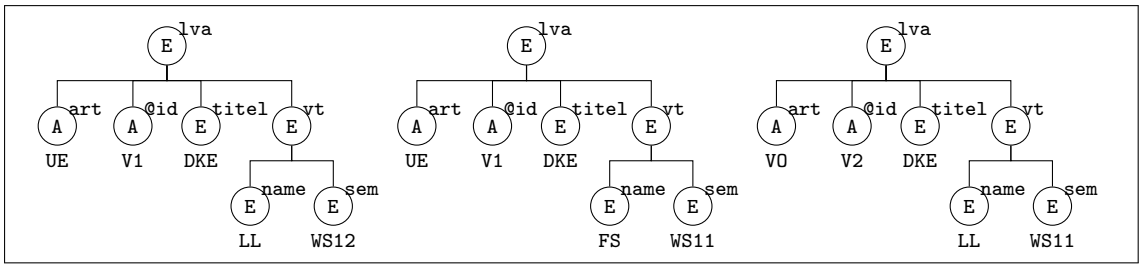

Abbildung 12.10: XML Schema Baum für die XML Bäume in den Abbildungen 4.29 und 4.30

Kontrollbeispiel 10.3 die drei in Abbildung 12.10 gezeigten Subgraphen.

Aus dieser Abbildung wird bereits ersichtlich, dass nur semantisch korrekte Kombinationen erzeugt werden. Die Funktionale Abhängigkeit ist – wie im Kontrollbeispiel 10.3. gefordert – erfüllt.

Ebenfalls richtig umgesetzt wird das Kontrollbeispiel 10.4. Hierbei ist der Constraint nicht erfüllt.

Da beide Kontrollbeispiele korrekt umgesetzt werden können, ist das Vergleichsmerkmal 10 erfüllt.

12.3 Zusammenfassung

In Abbildung 12.11 wird das Ergebnis nochmals zusammengefasst.

Der Ansatz von Hartmann, Link und Kirchberg geht davon aus, dass ein XML Schema Baum vorhanden ist. Das Vergleichsmerkmal 1 ist daher nicht erfüllt. Selbiges gilt für das Vergleichsmerkmal 2, da beim Subgraph-basierten Ansatz nicht mittels Pfaden auf die Knoten bzw. Werte zugegriffen wird. Betreffend dem Baummodell wird weder Vergleichsmerkmal 3 noch 4 erfüllt, da die Knoten nicht geordnet sein müssen und gemischter Inhalt in XML Dokumenten nicht unterstützt wird. Erfüllt ist hingegen das Vergleichsmerkmal 5, weil eine Definition von n-ären Constraints ermöglicht wird.

Da gemischter Inhalt nicht erlaubt ist, kann auch das 6. Vergleichsmerkmal nicht erfüllt werden. Im Gegensatz dazu wird ein baumbasierter Wertvergleich ermöglicht und das Vergleichsmerkmal 7 ist dadurch erfüllt. Der Geltungsbereich der Constraints kann nicht eingeschränkt werden, sodass Vergleichsmerkmal 8 nicht erfüllt ist. Da Subgraphen mit nicht vorhandenen Datenobjekten bei der Prüfung von Constraints ignoriert werden, ist

Vergleichsmerkmale		
1	Schema	nicht erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	nicht erfüllt
4	Baummodell: Gemischter Inhalt	nicht erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	erfüllt

Abbildung 12.11: Vergleichsmerkmale für den Ansatz von Hartmann, Link und Kirchberg

eine semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit nicht vorhandenen Datenobjekten nicht garantiert. Die Vergleichsmerkmale 9.1 und 9.2 sind demnach nicht erfüllt. Im Gegensatz dazu wird das Vergleichsmerkmal 10 erfüllt, da eine semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten ermöglicht wird.

Kapitel 13

Ansatz von Mok

Als nächstes wird der Ansatz von Mok [64] vorgestellt. Es handelt sich dabei um einen Ansatz zur Definition von Funktionalen Abhängigkeiten mit Hilfe von Templates.

13.1 Kurzbeschreibung

Eine XTFD (XML Template Functional Dependency) besteht aus einer Hypothese und einer Konklusion. Die Hypothese beinhaltet eine endliche (möglicherweise leere) Liste von Variablen (Element-, Attribut- oder Textvariablen). In der Konklusion werden zwei Attribut- oder Textvariablen der in der Hypothese angegebenen Liste definiert, deren Werte verglichen werden. Als Voraussetzung dafür wird angegeben, dass das XML Dokument einer DTD oder einem XML Schema entsprechen muss. Das nachfolgende Beispiel soll die Vorgehensweise bei der Definition von XTFDs verdeutlichen. Dazu wird die in Abbildung 13.1 angegebene DTD verwendet.

Beispiel 13.1: *Eine XTFD soll definieren, dass der Titel einer Lehrveranstaltung deren ID bestimmt. Die XTFD lautet demnach*

$lva_1.titel_1$

$lva_1.id_1$

$lva_2.titel_1$

$lva_2.id_2$

$id_1 = id_2$

In der Hypothese sind die Variablen $lva_1.titel_1$, $lva_1.id_1$, $lva_2.titel_1$ *und*

$lva_2.id_2$ aufgelistet. In der Konklusion, welche sich unterhalb der waagrechten Linie befindet, werden zwei Variablen der Hypothese definiert, die gleich sein müssen. In diesem Beispiel sind dies die beiden Variablen id_1 und id_2. Die Funktionale Abhängigkeit besagt somit, dass, wenn zwei (nicht unbedingt verschiedene) lva-Elemente dieselben titel-Werte haben, auch deren id-Werte gleich sein müssen.

```
<!ELEMENT uni (lvas)>
<!ELEMENT lvas (lva*)>
<!ELEMENT lva (titel)>
<!ATTLIST lva id CDATA #REQUIRED>
<!ELEMENT titel (#PCDATA)>
```

Abbildung 13.1: DTD zur Darstellung der Definition von XTFDs

Neben der Definition von XTFDs stellt Mok auch Regeln für Attribut- und Textwerte vor, um Redundanzen zu entfernen. Des Weiteren vergleicht er seinen Ansatz mit dem *Tree Tuple* Ansatz von Arenas und Libkin [6, 7].

13.2 Überprüfung der Vergleichsmerkmale

Im Folgenden wird analysiert, ob der Ansatz von Mok die Vergleichsmerkmale in Kapitel 4 erfüllt.

13.2.1 Unabhängigkeit von einer Schemasprache

Wie bereits erwähnt wurde, setzt Mok zur Definition der XTFDs voraus, dass eine DTD oder ein XML Schema vorhanden ist und, dass das XML Dokument diesem Schema entsprechen muss. Der Ansatz ist also an das Vorhandensein einer DTD oder eines XML Schemas gebunden und Vergleichsmerkmal 1 ist daher nicht erfüllt.

13.2.2 Pfadsprache

Mok definiert in seinem Ansatz keine eigene Pfadsprache. Vielmehr definiert er XTFDs auf der Grundlage eines vorhandenen Schemas. Das Vergleichsmerkmal 2 ist somit nicht erfüllt.

13.2.3 Baummodell

In seinem Ansatz geht Mok nicht auf die Dokumentordnung ein. Vielmehr konzentriert er sich auf die Definition von Funktionalen Abhängigkeiten in datenorientierten XML Dokumenten, in denen die Dokumentordnung in der Regel nicht von Bedeutung ist (siehe Abschnitt 4.1.3). Das Vergleichsmerkmal 3 ist demnach nicht erfüllt.

Ebenfalls nicht erfüllt ist das Vergleichsmerkmal 4, da definiert wird, dass ein Elementknoten höchstens einen Textwert beinhalten kann und gemischter Inhalt somit nicht erlaubt ist.

13.2.4 Arität

Mok definiert, dass in der Hypothese eine endliche (möglicherweise leere) Menge von XTFD-Variablen angegeben werden kann, wobei diese folgende Form hat (informell):

- $\tau_1.\tau_2.\cdots.\tau_m$, mit τ_i, $m \leq 1$, als Elementvariable,
- $\tau_1.\tau_2.\cdots.\tau_m.a_j$ mit τ_i, $m \leq 1$, als Elementvariable und a_j als Attributvariable,
- $\tau_1.\tau_2.\cdots.\tau_m$, mit τ_i, $m \leq 1$, als Elementvariable und a_j als Stringvariable.

Wird diese Definition nun für die Kontrollbeispiele 5.3 und 5.4 verwendet, ergibt sich folgende XTFD:

$\texttt{vt}_1\texttt{.vn}_1$
$\texttt{vt}_1\texttt{.nn}_1$
$\texttt{vt}_1\texttt{.id}_1$
$\texttt{vt}_2\texttt{.vn}_1$
$\texttt{vt}_2\texttt{.nn}_1$
$\texttt{vt}_2\texttt{.id}_2$

$\texttt{id}_1$ = id $_2$

Für den XML Baum in Abbildung 4.9 ist der Constraint erfüllt, da immer wenn die Werte der Attribute `@vn` und `@nn` zweier Vortragenden übereinstimmen auch deren `@id`-Wert gleich ist. Das Kontrollbeispiel 5.3 wird somit korrekt umgesetzt.

Für das Kontrollbeispiel 5.4 soll diese XTFD für den XML Baum in Abbildung 4.10 verwendet werden. Hierbei sind die Werte für die Attribute `@vn` und `@nn` zweier Vortragen-

den identisch, deren `@id`-Werte unterscheiden sich jedoch. Die Funktionale Abhängigkeit wird demnach nicht erfüllt. Da dies dem gewünschten Ergebnis entspricht, wird das Kontrollbeispiel 5.4 ebenfalls korrekt umgesetzt und das Vergleichsmerkmal 5 ist demnach erfüllt.

13.2.5 Wertvergleich

Mok definiert in seinem Ansatz den Vergleich zweier Elementknoten anhand ihrer Identität. Attribut- und Textknoten werden hingegen anhand ihrer Stringwerte verglichen.

Ein Wertvergleich wird bei Elementknoten somit nicht ermöglicht und die Vergleichsmerkmale 6 und 7 sind nicht erfüllt.

13.2.6 Geltungsbereich

In diesem Ansatz wird keine Möglichkeit geboten, den Geltungsbereich eines Constraints einzuschränken. Aus diesem Grund ist das Vergleichsmerkmal 8 nicht erfüllt.

13.2.7 Nicht vorhandene Datenobjekte

Die Erfüllbarkeit der XTFDs wird in diesem Ansatz lediglich für jene *Selector*-Knoten getestet, deren *Field*-Knoten, d.h. Variablen, vorhanden sind. Fehlen einzelne *Field*-Knoten, werden die jeweiligen *Selector*-Knoten beim Vergleich ignoriert.

Für das Kontrollbeispiel 9.4 und den XML Baum in Abbildung 4.25 würde dies bedeuten, dass der Constraint lediglich für den ersten Vortragenden getestet wird, dessen *Field*-Knoten `name, lva` und `inst` vorhanden sind. Da für den zweiten Vortragenden kein `inst`-Wert vorhanden ist, wird dieser ignoriert. Die Funktionale Abhängigkeit ist demnach erfüllt. Dies entspricht sowohl unter der Annahme von *dne* als auch *unk* dem richtigen Ergebnis.

Ebenfalls aus gleichem Grund ist der Constraint für den XML Baum in Abbildung 4.26 erfüllt, da auch hier wiederum nur die Werte der ersten Vortragenden verwendet werden, deren ID eindeutig bestimmt werden kann. Dies liefert unter der Annahme von *dne* das richtige Ergebnis, nicht jedoch unter der Annahme von *unk*.

Für die `vt`-Elemente im XML Baum in Abbildung 4.27 wird überhaupt kein Vergleich durchgeführt, da für beide Vortragenden kein `inst`-Wert vorhanden ist. Eine semantisch

korrekte Umsetzung wird somit nicht ermöglicht. Die Vergleichsmerkmale 9.1 und 9.2 sind daher nicht erfüllt.

13.2.8 Mehrfach vorhandene Datenobjekte

Mok ermöglicht mit seinem Ansatz die Definition von XTFDs in XML Dokumenten mit mehrfach vorhandenen Datenobjekten. Wie dies umgesetzt wird, wird nun anhand der Kontrollbeispiele 10.3 und 10.4 getestet. Dafür wird folgende XTFD definiert:

$\texttt{lva}_1\texttt{.titel}_1\texttt{.S}_1$
$\texttt{lva}_1\texttt{.vt}_1\texttt{.name}_1\texttt{.S}_1$
$\texttt{lva}_1\texttt{.vt}_1\texttt{.sem}_1\texttt{.S}_1$
$\texttt{lva}_1\texttt{.id}_1$
$\texttt{lva}_2\texttt{.titel}_2\texttt{.S}_1$
$\texttt{lva}_2\texttt{.vt}_2\texttt{.name}_2\texttt{.S}_1$
$\texttt{lva}_2\texttt{.vt}_2\texttt{.sem}_2\texttt{.S}_1$
$\texttt{lva}_2\texttt{.id}_2$

$\texttt{id}_1 = \texttt{id}_2$

Diese XTFD besagt, dass, wenn zwei (nicht zwingend verschiedene) `lva`- und `vt`-Paare dieselben Werte für die Elemente `titel`, `name` und `sem` haben, auch ihre `id`-Werte gleich sein müssen.

Da Elementknoten anhand ihrer Identität verglichen werden, werden zur besseren Darstellung der Kontrollbeispiele wiederum die mit Knotennummern versehene XML Bäume in den Abbildungen 10.4 und 10.5 (siehe Abschnitt 10.2.8) verwendet.

Wird die XTFD nun für den XML Baum in Abbildung 10.4 getestet, führt dies zu folgendem Ergebnis:

$\texttt{lva}_1$	$\texttt{titel}_1.\texttt{S}_1$	$\texttt{vt}_1$	$\texttt{name}_1.\texttt{S}_1$	$\texttt{sem}_1.\texttt{S}_1$	$\texttt{id}_1$
$\texttt{lva}_2$	$\texttt{titel}_2.\texttt{S}_1$	$\texttt{vt}_2$	$\texttt{name}_2.\texttt{S}_1$	$\texttt{sem}_2.\texttt{S}_1$	$\texttt{id}_2$
v_2	DKE	v_6	Laura Leitner	WS12	V1
v_2	DKE	v_6	Laura Leitner	WS12	V1
v_2	DKE	v_9	Felix Schnell	WS11	V1
v_2	DKE	v_9	Felix Schnell	WS11	V1
v_{12}	DKE	v_{16}	Laura Leitner	WS11	V2
v_{12}	DKE	v_{16}	Laura Leitner	WS11	V2

Diese Auflistung verdeutlicht, dass die XTFD für den XML Baum in Abbildung 10.4 korrekt umgesetzt werden kann, weil die IDs der Lehrveranstaltungen eindeutig bestimmt werden können und der Constraint, wie im Kontrollbeispiel 10.3 gefordert, erfüllt ist.

Bei der Definition der XTFD ist zu beachten, dass eine leicht veränderte Definition zu einem falschen Ergebnis führen könnte [49]. Würden beispielsweise mehrere verschiedenen `vt`-Elemente innerhalb eines `lva`-Elements angegeben werden, würde diese zu einem falschen Ergebnis führen. Die semantisch korrekte Umsetzung ist daher von einer genauen Definition des Constraints durch den Entwickler abhängig [49].

Wird die XTFD nun für das Kontrollbeispiel 10.4 und den XML Baum in Abbildung 10.5 verwendet, ergibt sich Folgendes:

$\texttt{lva}_1$	$\texttt{titel}_1.\texttt{S}_1$	$\texttt{vt}_1$	$\texttt{name}_1.\texttt{S}_1$	$\texttt{sem}_1.\texttt{S}_1$	$\texttt{id}_1$
$\texttt{lva}_2$	$\texttt{titel}_2.\texttt{S}_1$	$\texttt{vt}_2$	$\texttt{name}_2.\texttt{S}_1$	$\texttt{sem}_2.\texttt{S}_1$	$\texttt{id}_2$
v_2	DKE	v_6	Laura Leitner	WS12	V1
v_2	DKE	v_6	Laura Leitner	WS12	V1
v_2	DKE	v_9	Felix Schnell	WS11	V1
v_{12}	DKE	v_{16}	Felix Schnell	WS11	V2

Hier ist die Funktionale Abhängigkeit nicht erfüllt, wie im unteren Bereich ersichtlich wird. Die Werte der Elemente `titel, name` und `sem` sind für $\texttt{lva}_1$ und $\texttt{vt}_1$ sowie $\texttt{lva}_2$ und $\texttt{vt}_2$ gleich, die `id`-Werte unterscheiden sich jedoch. Da dies gefordert ist, wird das Kontrollbeispiel 10.4 korrekt umgesetzt und das Vergleichsmerkmal 10 ist erfüllt.

13.3 Zusammenfassung

Abbildung 13.2 fasst die Ergebnisse der Analyse dieses Ansatzes nochmals zusammen.

Vergleichsmerkmale		
1	Schema	nicht erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	nicht erfüllt
4	Baummodell: Gemischter Inhalt	nicht erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	nicht erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	nicht erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	erfüllt

Abbildung 13.2: Vergleichsmerkmale für den Ansatz von Mok

Da Mok in seinem Ansatz voraussetzt, dass das XML Dokument einer DTD oder einem XML Schema entsprechen muss, ist das 1. Vergleichsmerkmal nicht erfüllt. Des Weiteren wird keine Pfadsprache definiert, sodass auch das 2. Vergleichsmerkmal nicht erfüllt ist. Mok gibt keine Definition eines Baummodells an, sodass die Vergleichsmerkmale 3 (Dokumentordnung) und 4 (Gemischter Inhalt) ebenfalls nicht erfüllt sind.

Erfüllt ist hingegen das 5. Vergleichsmerkmal, da mit der Definition der Hypothese einer XTFD als Menge von Variablen n-äre Constraints ermöglicht werden.

Mok beschränkt den Wertvergleich auf Attribut- und Textwert. Die Vergleichsmerkmale 6 (Wertvergleich durch Verkettung von Textwerten) und 7 (baumbasierter Wertvergleich) sind somit nicht erfüllt. Selbiges gilt für das Vergleichsmerkmal 8, da keine Einschränkung des Geltungsbereichs möglich ist.

Bei der Überprüfung des Ansatz hinsichtlich der Vorgehensweise bei XML Dokumenten mit nicht vorhandenen Datenobjekten stellt sich heraus, dass eine semantisch korrekte Umsetzung nicht ermöglicht wird. Der Grund dafür ist, dass Mok nur jene *Selector*-Knoten vergleicht, deren *Field*-Knoten alle vorhanden sind. *Selector*-Knoten mit einer unvollständigen Menge von *Field*-Knoten werden ignoriert, sodass die Vergleichsmerkmale 9.1 und 9.2 nicht erfüllt sind. Im Gegensatz dazu ist das 10. Vergleichsmerkmal erfüllt, da eine semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten ermöglicht wird.

Kapitel 14

Ansatz von Karlinger

Karlinger [49] präsentiert einen Ansatz zur Definition von Schlüsseln, Fremdschlüsseln und Inklusionsabhängigkeiten. Dieser wird im Folgenden beschrieben.

14.1 Kurzbeschreibung

Karlinger definiert einen Schlüssel folgendermaßen: XKey = $(S, \{F_1, \cdots, F_n\})$, wobei *S* den *Selector* definiert und $\{F_1, \cdots, F_n\}$ steht für eine nicht leere Menge von Pfaden, die mit einem Attribut- oder Textknoten enden.

Inklusionsabhängigkeiten werden definiert als XIND = $(S, [F_1, \cdots, F_n]) \subseteq (S', [F'_1, \cdots, F'_n])$, wobei S und S' *Selector*-Pfade sind die mit Elementknoten enden und $[F_1, \cdots, F_n]$ und $[F'_1, \cdots, F'_n]$ sind nicht leere Listen von Pfaden, welche wiederum mit Attribut- oder Textknoten enden.

Neben der Definition von Schlüsselbedingungen und Inklusionsabhängigkeiten präsentiert Karlinger eine Methode, flache relationale Datenbanken in XML Dokumente umzuwandeln und gleichzeitig die relationale Semantik mit Hilfe der XKeys und XINDs zu erhalten. Des Weiteren werden Ergebnisse zum Reasoning vorgestellt.

14.2 Überprüfung der Vergleichsmerkmale

In diesem Abschnitt wird der Ansatz von Karlinger mit Hilf der Vergleichsmerkmale in Kapitel 4 analysiert.

14.2.1 Unabhängigkeit von einer Schemasprache

Karlinger bindet seinen Ansatz an keine Schemasprache und setzt auch kein Vorhandensein einer Schemaspezifikation voraus. Das 1. Vergleichsmerkmal ist daher erfüllt.

14.2.2 Pfadsprache

Pfade werden definiert als $P = l_1.\cdots.l_n$, wobei $l_i, i \in \{1, \cdots, n-1\}$, für einen Elementknoten steht und l_n ist ein Element-, Attribut- oder Textknoten. Obwohl generell erlaubt ist, dass ein Pfad mit einem Elementknoten endet, wird dies für *Field*-Pfade jedoch nicht ermöglicht, da diese immer mit Attribut- oder Textknoten enden müssen. Des Weiteren wird der Kleene Operator nicht unterstützt, sodass der Schlüssel σ = `(_*.vt, {name})` und die Inklusionsabhängigkeit σ = `(_*.lva.vt, [name])` $\subseteq$ `(_*.vortragende.vt, [name])` für die Kontrollbeispiele 2.1 und 2.2 bzw. 2.5 und 2.6 nicht definiert werden können. Das Vergleichsmerkmal 2 ist somit nicht erfüllt.

14.2.3 Baummodell

Karlinger geht in seinem Ansatz von einem geordneten Baum aus. Dazu verwendet er die Funktion $\leq$, d.h. $v_1 \leq v_2$ bedeutet, dass der Knoten v_1 vor dem Knoten v_2 im Baum steht. Die Dokumentordnung wird somit eingehalten und der XML Baum im Kontrollbeispiel 3.1 kann abgebildet werden. Das Vergleichsmerkmal 3 ist daher erfüllt.

Des Weiteren wird erlaubt, dass Elementknoten mehrere Textknoten beinhalten können, sodass gemischter Inhalt möglich ist. Der XML Baum im Kontrollbeispiel 4.1 kann daher korrekt abgebildet werden und das Vergleichsmerkmal 4 ist ebenfalls erfüllt.

14.2.4 Arität

Durch die Definition der *Fields* als eine nicht leere Menge bzw. Liste von Pfaden wird bereits deutlich, dass auch n-äre Constraints definiert werden können.

Wird dies anhand der Kontrollbeispiele 5.1 und 5.2 bzw. 5.5 und 5.6 und dem Schlüssel σ = `(uni.vortragende.vt, {@vn, @nn})` sowie der Inklusionsabhängigkeit σ = `(uni.lvas.lva.vt, [@vn, @nn])` $\subseteq$ `(uni.vortragende.vt, [@vn, @nn])` getestet, stellt sich heraus, dass diese definiert werden können und zum korrekten Ergebnis führen. Das Vergleichsmerkmal 5 ist demnach erfüllt.

14.2.5 Wertvergleich

Um den Wert eines Knotens zu definieren, verwendet Karlinger die Funktion $val(v)$. Sie gibt für Attribut- und Textknoten deren Stringwert zurück. Für Elementknoten gibt es jedoch keine Definition der Wertgleichheit. Aus diesem Grund, und auch weil nur Attribut- und Textknoten als *Field*-Knoten erlaubt sind, können die Kontrollbeispiele der Vergleichsmerkmale 6 und 7 nicht umgesetzt werden und sind daher nicht erfüllt.

14.2.6 Geltungsbereich

Mit der von Karlinger präsentierten Definition der Schlüssel und Inklusionsabhängigkeiten wird keine Einschränkung des Geltungsbereichs ermöglicht. Der Schlüssel σ = `(uni.institut.vortragende.vt, (lva, {@titel}))` und die Inklusionsabhängigkeit σ = `(uni.institut, (vortragende.vt.lva, [@titel])` $\subseteq$ `(lvas.lva, [@titel]))` der Kontrollbeispiele 8.1 und 8.2 bzw. 8.5 und 8.6 können somit nicht definiert werden. Aus diesem Grund wird das Vergleichsmerkmal 8 nicht erfüllt.

14.2.7 Nicht vorhandene Datenobjekte

Karlinger geht sehr genau auf die Definition von Integritätsbedingungen in XML Dokumenten mit nicht vorhandenen Datenobjekten ein. Er geht dabei von *dne* und *no info* aus. Da *unk* vernachlässigt wird, ist das Vergleichsmerkmal 9.2 nicht erfüllt.

Um eine semantisch korrekte Umsetzung unter der Annahme von *dne* und *no info* zu gewährleisten, definiert Karlinger die sogenannte *Maximum Combination of Field Nodes*:

Definition 14.1: *"Let S be a selector and let* $\{F_1, \cdots, F_n\}$ *be a set of fields. Also, let T be an XML tree and let* $v \in$ *nodes(S, T)*[16] *be a selector node. A set of nodes* $\{v_1, \cdots, v_m\}$ *in XML tree T is defined to be a maximum combination of field nodes for selector node v with respect to the fields* $\{F_1, \cdots, F_n\}$ *if*

1. $\forall\{i, j\} \subseteq \{1, \cdots, m\}$, *closest*$(v_i, v_j) = true$;
2. $\forall i \in \{1, \cdots, m\}$, $v \in$ *ancestor* (v_i) *and* $v_i \in$ *nodes(S.*F_j*) for some* $j \in \{1, \cdots, n\}$;

[16] Die Funktion *nodes(S, T)* gibt eine Menge von Knoten des XML Baums *T* zurück, welche vom Wurzelknoten über den Pfad *P* erreichbar ist.

3. *there does not exist node* $\bar{v} \in$ *desc(v) such that* $\forall i \in \{1, \cdots, m\}, \bar{v} \neq v_i$ *and closest* $(\bar{v}, v_i)$ *= true and* $\bar{v} \in nodes(S.F_j)$ *for some* $j \in \{1, \cdots, n\}$."

Die *Closest*-Eigenschaft entspricht dabei der im Ansatz von Vincent, Liu und Mohania [85] vorgestellten Definition (siehe Abschnitt 10). Zwei Knoten erfüllen demnach die *Closest*-Eigenschaft, wenn sie einen gemeinsamen Ancestor-or-Self Knoten haben, der sich aus der Schnittmenge jener Pfade ergibt, die zu den jeweiligen Knoten führen. Die *Maximum Combination* wird von einer Menge von *Field*-Knoten erfüllt, wenn sie paarweise die *Closest*-Eigenschaft erfüllen, denselben *Selector*-Knoten haben und es keine größere Menge von semantisch richtigen Kombinationen der *field*-Knoten gibt. Ebenso wie im Ansatz von Vincent, Liu und Mohania [85] müssen die *Field*-Knoten somit untereinander die *Closest*-Eigenschaft erfüllt und auch zum *Selector*-Knoten.

Um herauszufinden, ob mit Hilfe der *Maximum Combination* eine semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit nicht vorhandenen Datenobjekten garantiert wird, werden die Kontrollbeispiele 9.1, 9.2 und 9.3 bzw. 9.7 und 9.8 verwendet. Um das Ergebnis besser veranschaulichen zu können, werden die im Abschnitt 10 abgebildeten XML Bäume der Kontrollbeispiele mit Nummerierung der Knoten verwendet.

Für die Kontrollbeispiele 9.1, 9.2 und 9.3 soll der Schlüssel σ = `(uni.vortragende.vt, {name.S, lva.S, inst.S})` definiert werden. Der XML Baum in Abbildung 10.1 zeigt, dass das zweite Element `vt` im Teilbaum `vortragende` kein Unterelement `inst` beinhaltet. Es ergeben sich daher für den *Selector*-Knoten v_2 die *Field*-Knoten $\{v_4, v_5, v_6\}$ und für v_7 die *Field*-Knoten $\{v_9, v_{10}\}$. Der Constraint ist hierbei erfüllt, weil es keine zwei Mengen von *Field*-Knoten gibt, die die *Maximum Combination* Eigenschaft erfüllen, und deren Werte und Anzahl der Knoten gleich sind. Das Kontrollbeispiel 9.1 wurde somit unter der Annahme von *dne* korrekt umgesetzt.

Selbiges gilt für das Kontrollbeispiel 9.2 und den XML Baum in Abbildung 10.2. Auch hier ist der Constraint erfüllt, weil sich die *Field*-Knoten $\{v_4, v_5, v_6\}$ von *Selector* v_2 und die *Field*-Knoten $\{v_9, v_{10}\}$ von *Selector* v_7 in Anzahl und Wert unterscheiden, und die Vortragenden eindeutig identifiziert werden können.

Ebenfalls richtig umgesetzt wird das Kontrollbeispiel 9.3 für den XML Baum in Abbildung 10.3. Hier ist der Constraint nicht erfüllt, weil die *Maximum Combinations* der *Field*-Knoten gleich sind. Für den *Selector* v_2 ist dies die Menge $\{v_4, v_5\}$ und für den *Selector* v_6 die Menge $\{v_8, v_9\}$. Diese stimmen in Anzahl und Wert überein und sind somit gleich,

sodass die Vortragenden nicht eindeutig identifiziert werden können.

Das Vergleichsmerkmal 9.1 ist somit für Schlüsselbedingungen erfüllt.

Für die Kontrolle der Inklusionsabhängigkeiten soll der Constraint σ = (uni.lvas.lva, [vt.S, titel.S, inst.S]) $\subseteq$ (vortragende.vt, [name.S, lva.S, inst.S]) definiert werden. Für den XML Baum in Abbildung 10.1 ergeben sich dabei die *Field*-Knoten $\{v_{13}, v_{14}\}$ auf der LHS und $\{v_4, v_5, v_6\}$ sowie $\{v_9, v_{10}\}$ auf der RHS. Da sowohl die Anzahl als auch die Werte von $\{v_{13}, v_{14}\}$ mit $\{v_9, v_{10}\}$ übereinstimmen, ist die Inklusionsabhängigkeit erfüllt und wie im Kontrollbeispiel 9.7 gefordert korrekt umgesetzt.

Ebenfalls richtig umgesetzt wird das Kontrollbeispiel 9.8. Der Constraint ist hierbei nicht erfüllt, weil die *Field*-Knoten $\{v_9, v_{10}\}$ der LHS mit den *Field*-Knoten $\{v_{13}, v_{14}\}$ der RHS zwar in der Anzahl übereinstimmen, deren Werte jedoch unterschiedlich sind. Die Inklusionsabhängigkeit ist daher nicht erfüllt.

Da beide Kontrollbeispiele korrekt umgesetzt werden können, ist das Vergleichsmerkmal 9.1 auch für Inklusionsabhängigkeiten erfüllt.

14.2.8 Mehrfach vorhandene Datenobjekte

Für die Definition von Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten wird die bereits vorgestellte *Closest*-Eigenschaft verwendet. Um herauszufinden, ob eine semantisch korrekte Umsetzung mit diesem Ansatz möglich ist, werden die Kontrollbeispiele 10.1 und 10.2 bzw. 10.5 und 10.6 verwendet. Die dafür verwendeten XML Bäume wurden mit nummerierten Knoten bereits im Abschnitt 10.2.8 dargestellt.

Zur Kontrolle der Schlüsselbedingungen wird der Constraint σ = (uni.lvas.lva, {titel.S, vt.name.S, vt.sem.S}) verwendet. Dabei ergeben sich als *Field*-Knoten für den XML Baum in Abbildung 10.4 die Mengen $\{v_5, v_7, v_8\}$, $\{v_5, v_{10}, v_{11}\}$ und $\{v_{12}, v_{17}, v_{18}\}$. Diese stimmen zwar in ihrer Anzahl überein, deren Werte sind jedoch verschieden. Deutlich wird dabei, dass mit Hilfe der *Closest*-Funktion nur semantisch korrekte Wertekombinationen gebildet werden. Die lva-Elemente können demnach eindeutig identifiziert werden und das Kontrollbeispiel 10.1 wird somit richtig umgesetzt.

Wird der Constraint für den XML Baum in Abbildung 10.5 verwendet, ist dieser nicht erfüllt. Es ergeben sich dabei die *Field*-Knoten $\{v_5, v_7, v_8\}$, $\{v_5, v_{10}, v_{11}\}$ und $\{v_{12}, v_{17}, v_{18}\}$. Die Anzahl und Werte von $\{v_5, v_7, v_8\}$ und $\{v_{12}, v_{17}, v_{18}\}$ stimmen überein, sodass die Vortragenden nicht eindeutig identifiziert werden können. Der Constraint ist somit, wie

gefordert, nicht erfüllt. Da beide Kontrollbeispiele korrekt umgesetzt wurden, ist das Vergleichsmerkmal 10 für Schlüsseln erfüllt.

Als nächstes soll die Vorgehensweise für Inklusionsabhängigkeiten getestet werden. Für den Constraint σ = `(uni.vortragende.vt [name.S, lva.titel.S, lva.sem.S])` $\subseteq$ `(uni.lvas.lva, [vt.name.S, titel.S, vt.sem.S])` ergeben sich im XML Baum in Abbildung 10.4 die *Field*-Knoten $[v_{21}, v_{24}, v_{25}]$ als LHS und $[v_7, v_5, v_8]$, $[v_{10}, v_5, v_{11}]$ und $[v_{17}, v_{15}, v_{18}]$ als RHS. Die Anzahl der Knoten und die Werte der *Field*-Knoten $[v_{21}, v_{24}, v_{25}]$ stimmen mit den *Field*-Knoten $[v_7, v_5, v_8]$ überein, sodass die Inklusionsabhängigkeit, wie in Kontrollbeispiel 10.5 gefordert, erfüllt ist.

Für das Kontrollbeispiel 10.6 soll der Constraint für den XML Baum in Abbildung 10.5 getestet werden. Die *Field*-Knoten der LHS sind dabei $[v_{21}, v_{24}, v_{25}]$. Diese stimmen zwar in der Anzahl, nicht aber in deren Werte mit den *Field*-Knoten $[v_7, v_5, v_8]$, $[v_{10}, v_5, v_{11}]$ oder $[v_{17}, v_{15}, v_{18}]$ der RHS überein. Der Constraint ist daher nicht erfüllt.

Beide Vergleichsmerkmale wurden somit korrekt umgesetzt und das Vergleichsmerkmal 10 ist auch für Inklusionsabhängigkeiten erfüllt.

14.3 Zusammenfassung

Abbildung 14.1 fasst das Ergebnis nochmals zusammen. Der Ansatz kann unabhängig von einer Schemasprache definiert werden. Das 1. Vergleichsmerkmal ist daher erfüllt. Vergleichsmerkmal 2 ist jedoch nicht erfüllt, weil der Kleene Operator nicht unterstützt wird und die Kontrollbeispiele auch aufgrund der Einschränkung, dass *Field*-Knoten immer Attribut- oder Textknoten sein müssen, nicht umgesetzt werden können.

Im Baummodell definiert Karlinger, dass die Dokumentordnung eingehalten werden muss und gemischter Inhalt erlaubt sind. Die beiden Vergleichsmerkmale 3 und 4 sind demnach erfüllt. Ebenfalls erfüllt ist das 5. Vergleichsmerkmal, da aufgrund der Definition der *Fields* als Menge bzw. Liste von Pfaden, n-äre Constraints ermöglicht werden.

Der Wertvergleich von Knoten wird auf Attribut- und Textknoten beschränkt, sodass keine Möglichkeit geboten wird, Elementwerte zu vergleichen. Die Vergleichsmerkmale 6 (Wertvergleich durch Verkettung von Textwerten) und 7 (Baumbasierter Wertvergleich) sind daher nicht erfüllt. Auch eine Einschränkung des Geltungsbereichs wird nicht ermöglicht und das Vergleichsmerkmal 8 ist ebenfalls nicht erfüllt.

Bei der Umsetzung von Constraints in XML Dokumenten mit nicht vorhandenen Daten-

Vergleichsmerkmale		
1	Schema	erfüllt
2	Pfadsprache	nicht erfüllt
3	Baummodel: Dokumentordnung	erfüllt
4	Baummodell: Gemischter Inhalt	erfüllt
5	Arität	erfüllt
6	Wertvergleich von gemischtem Inhalt	nicht erfüllt
7	Baumbasierter Wertvergleich	nicht erfüllt
8	Geltungsbereich	nicht erfüllt
9.1	Nicht vorhandene Datenobjekte: *dne*	erfüllt
9.2	Nicht vorhandene Datenobjekte: *unk*	nicht erfüllt
10	Mehrfach vorhandene Datenobjekte	erfüllt

Abbildung 14.1: Vergleichsmerkmale für den Ansatz von Karlinger

objekten geht Karlinger von *dne* und *no info* aus. Hierbei wird eine semantisch korrekte Umsetzung garantiert. Unter der Annahme von *unk* ist dies jedoch nicht der Fall. Das Vergleichsmerkmal 9.1 ist demnach erfüllt, Vergleichsmerkmal 9.2 jedoch nicht. Bei der Definition von Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten wird mit der *Closest*-Eigenschaft eine Möglichkeit geboten, semantisch korrekte Kombinationen der Werte zu garantieren. Vergleichsmerkmal 10 ist daher erfüllt.

Kapitel 15

Zusammenfassung

In diesem Kapitel werden die im Zuge dieser Arbeit ausgearbeiteten Ergebnisse nochmals zusammengefasst.

15.1 Resümee

Abbildung 15.1 zeigt die Ergebnisse der Analyse der einzelnen Ansätze.

Aus dieser Auflistung wird bereits ersichtlich, dass in allen Ansätzen eine Definition von n-ären Constraints ermöglicht wird.

Das genaue Gegenteil trifft für das 6. Vergleichsmerkmal zu, das den Wertvergleich von Elementknoten basierend auf der Verkettung aller direkt und indirekt nachfolgenden Textwerte betrifft. Es wird zwar in einigen Ansätzen das Vorhandensein von gemischtem Inhalt im Baummodell erlaubt (Ansätze von W3C [76], Buneman et al. [18, 19, 20], Fan und Libkin [34], Lee, Ling und Low [58], Vincent, Liu und Liu [83, 84], Vincent et al. [86] und Karlinger [49]), es wird jedoch keine Methode angegeben, wie dies als Wertvergleich in Constraints umgesetzt werden kann.

Ebenfalls von keinem Ansatz erfüllt wird das Vergleichsmerkmal 9.2, das die semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit nicht vorhandenen Datenobjekten unter der Annahme von *unk* betrifft.

Alle anderen Vergleichsmerkmale wurden von einzelnen oder mehreren Ansätzen erfüllt.

Die Ansätze von Buneman et al. [18, 19, 20], Vincent, Liu und Liu [83, 84], Vincent

et al. [86] und Karlinger [49] sind an keine Schemasprache gebunden und können somit unabhängig von einem Schema umgesetzt werden. Der Ansatz des W3C [76] ist auf XML Schema aufgebaut. Arenas und Libkin [6, 7] setzen voraus, dass eine DTD vorhanden sein muss. Selbiges gilt für den Ansatz von Fan und Libkin [34]. Lee, Ling und Low [58] definieren ihren Ansatz auf Grundlage einer DTD und der Ansatz von Hartmann, Link und Kirchberg [39, 43] geht vom Vorhandensein eines Schemabaumes aus. Der Ansatz von Mok [64] kann nur umgesetzt werden, wenn entweder eine DTD oder ein XML Schema vorhanden ist, dem das XML Dokument entsprechen muss.

Das 2. Vergleichsmerkmal wird nur von zwei Ansätzen erfüllt. Das W3C [76] verwendet zur Definition der Constraint XPath-Ausdrücke. Eine eigene Pfadsprache wurde von Buneman et al. [18, 19, 20] entwickelt, welche auch den Kleene Operator beinhaltet. In beiden Ansätzen wird es erlaubt, dass Pfade zu den *Field*-Knoten in Elementknoten enden.

Die Vergleichsmerkmale 3 und 4 betreffen das Baummodell. Das Einhalten der Dokumentordnung wird in den Ansätzen von W3C [76], Buneman et al. [18, 19, 20], Fan und Libkin [34], Lee, Ling und Low [58], Vincent, Liu und Liu [83, 84], Vincent et al. [86] und Karlinger [49] gefordert. Das Vorhandensein von gemischtem Inhalt in XML Dokumenten wird von denselben Ansätzen erlaubt.

Ermöglicht wird ein baumbasierter Wertvergleich in den Ansätzen von W3C [76], Buneman et al. [18, 19, 20] und Hartmann, Link und Kirchberg [39, 43].

Das 8. Vergleichsmerkmal analysiert die Möglichkeit zur Einschränkung des Geltungsbereichs eines Constraints. Dies wird nur in den Ansätzen von W3C [76] und Buneman et al. [18, 19, 20] ermöglicht.

Eine semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit nicht vorhandenen Datenobjekten wird lediglich von Karlinger [49] ermöglicht. Hierbei wird von *dne* und *no info* ausgegangen, sodass das Vergleichsmerkmal 9.1 erfüllt ist.

Das letzte Vergleichsmerkmal betrifft die Tatsache, dass mehrere Unterelemente mit demselben Label innerhalb eines Elements vorkommen können. Eine semantisch korrekte Umsetzung von Constraints in XML Dokumenten mit mehrfach vorhandenen Datenobjekten wird in den Ansätzen von Arenas und Libkin [6, 7], Vincent, Liu und Mohania [85], Hartmann, Link und Kirchberg [39, 43], Mok [64] und Karlinger [49] ermöglicht.

Abschließend wird nochmals darauf hingewiesen, dass es sich bei diesen Ansätzen le-

Quelle	1 Schemasprache	2 Pfadsprache	3 Dokumentordnung	4 Gemischter Inhalt	5 Arität	6 WV gemischter Inhalt	7 Baumbasierter WV	8 Geltungsbereich	9.1 Nicht vorhandene Datenobjekte - *dne*	9.2 Nicht vorhandene Datenobjekte - *unk*	10 Mehrfach vorhandene Datenobjekte
World Wide Web Cons.		✓	✓	✓	✓		✓	✓			
Buneman et al.	✓	✓	✓	✓	✓		✓	✓			
Arenas, Libkin					✓						✓
Fan, Libkin			✓	✓	✓						
Lee, Ling, Low			✓	✓	✓						
Vincent, Liu, Liu	✓		✓	✓	✓						
Vincent, Liu, Mohania	✓		✓	✓	✓						✓
Vincent et al.	✓		✓	✓	✓						
Hartmann, Link, Kirchberg					✓		✓				✓
Mok					✓						✓
Karlinger	✓		✓	✓	✓				✓		✓

Abbildung 15.1: Zusammenfassung aller verglichenen Ansätze

diglich um eine kleine Auswahl von Ansätzen handelt, welche sich teilweise grundlegend voneinander unterscheiden. Ein Vergleich aller bisher entwickelten Ansätze wäre im Zuge dieser Diplomarbeit nicht möglich gewesen, da es den Rahmen gesprengt hätte. Interessierte Leser werden auf Karlinger [49] verwiesen, der neben der Vorstellung eines eigenen Ansatzes auch einen Überblick zahlreicher bisher vorhandener Ansätze wertbasierter Integritätsbedingungen bietet.

15.2 Schlussbemerkung und Ausblick

Die im Zuge dieser Diplomarbeit entwickelte Liste von Vergleichsmerkmalen wurde für die Analyse von Ansätzen wertbasierter XML Integritätsbedingungen entwickelt. Darüber hinaus existieren noch weitere Kategorien von Integritätsbedingungen (siehe Karlinger [49]). Dies sind Schema-Constraints, welche mittels Schema Spezifikationen definiert werden können, Pfad-Constraints, die Datenobjekte über die Position im XML Baum definieren, und komplexe Constraints, die mit Hilfe von Regeln definiert werden. [49] Ein weiterer Schritt bei der Analyse und dem Vergleich von IC-Ansätze wäre daher, auch für diese Kategorien gemeinsame Merkmale heraus zu arbeiten und somit eine Vergleichsbasis zu finden.

Des Weiteren könnten, neben den im Abschnitt 4 vorgestellten Vergleichsmerkmalen, noch weitere Eigenschaften herangezogen werden. Diese betreffen beispielsweise die Ausdruckskraft der verwendeten Pfadsprache oder auch die Analyse des Reasonings.

Überdies kann die Analyse unter Verwendung der in Abschnitt 4 beschriebenen Vergleichsmerkmale beliebig auf weitere Ansätze wertbasierter IC-Ansätze ausgedehnt werden.

Abbildungsverzeichnis

2.1 Rahmenbeispiel 7

2.2 Baumdarstellung des Rahmenbeispiels 14

2.3 XML Baum zur Darstellung der Achsen 15

4.1 XML Baum zur Darstellung der Wichtigkeit von Kleene Operatoren . . . 34

4.2 XML Baum 1 zur Überprüfung der Pfadfunktionen 36

4.3 XML Baum 2 zur Überprüfung der Pfadfunktionen 36

4.4 Datenorientierte XML Dokumente mit unterschiedlicher Dokumentordnung 41

4.5 Dokumentorientierte XML Dokumente mit unterschiedlicher Dokumentordnung 42

4.6 XML Baum zur Überprüfung der Dokumentordnung 43

4.7 Wegbeschreibung in Form eines XHTML Dokuments 44

4.8 XML Baum zur Überprüfung von gemischtem Inhalt im Baummodell . . 45

4.9 XML Baum 1 zur Überprüfung der Arität 46

4.10 XML Baum 2 zur Überprüfung der Arität 46

4.11 XML Baum zur Darstellung von Knoten- und Wertgleichheit 50

4.12 XML Baum zur Gegenüberstellung von einfachem Wertvergleich und Wertvergleich basierend auf der Verkettung aller direkt und indirekt nachfolgenden Textwerte 51

4.13 XML Baum zur Gegenüberstellung vom Wertvergleich basierend auf der Verkettung von Textwerten und baumbasiertem Wertvergleich 52

4.14 XML Dokument zur Gegenüberstellung vom Wertvergleich basierend auf der Verkettung von Textwerten und baumbasiertem Wertvergleich 52

4.15 XML Baum 1 zur Überprüfung von gemischtem Inhalt in XML Constraints 53

4.16 XML Baum 2 zur Überprüfung von gemischtem Inhalt in XML Constraints 54

4.17 XML Baum 1 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Vortragende 57

4.18 XML Baum 1 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Lehrveranstaltungen 57

4.19 XML Baum 2 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Vortragende 58

4.20 XML Baum 2 zur Überprüfung des baumbasierten Wertvergleichs - Teilbaum Lehrveranstaltungen 58

4.21 XML Baum zum Unterscheiden zwischen Wertvergleich durch Verkettung von Textwerten sowie baumbasiertem Wertvergleich und n-ärem Wertvergleich 61

4.22 XML Baum 1 zur Überprüfung des Geltungsbereichs 63

4.23 XML Baum 2 zur Überprüfung des Geltungsbereichs 63

4.24 XML Baum zur Darstellung eines falschen Ergebnisses beim Ignorieren nicht vorhandener Datenobjekte 68

4.25 XML Baum 1 zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte 71

4.26 XML Baum 2 zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte 71

4.27 XML Baum 3 zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte 72

4.28 XML Baum zur Darstellung von richtigen Wertekombinationen bei mehrfach vorhandenen Datenobjekten 78

4.29 XML Baum 1 zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten 78

4.30 XML Baum 2 zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten 78

4.31 Vergleichsmerkmale 84

4.32 Related Work 89

5.1 Vergleichsmerkmale für den Ansatz vom World Wide Web Consortium . 105

6.1 Vergleichsmerkmale für den Ansatz von Buneman et al. 116

7.1 DTD zur Überprüfung der Kontrollbeispiele 10.3 und 10.4 123

7.2 Vergleichsmerkmale für den Ansatz von Arenas und Libkin 125

8.1 Vergleichsmerkmale für den Ansatz von Fan und Libkin 133

9.1 Aufbau einer Funktionalen Abhängigkeit als DTD dargestellt 136

9.2 Vergleichsmerkmale für den Ansatz von Lee, Ling und Low 141

10.1 XML Baum 1 mit beschrifteten Knoten zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte 147

10.2 XML Baum 2 mit beschrifteten Knoten zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte 148

10.3 XML Baum 3 mit beschrifteten Knoten zur Überprüfung der korrekten Umsetzung von Constraints mit nicht vorhandenen Datenobjekte 148

10.4 XML Baum 1 mit nummerierten Knoten zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten 151

10.5 XML Baum 2 mit nummerierten Knoten zur Überprüfung von Constraints mit mehrfach vorhandenen Datenobjekten 151

10.6 Vergleichsmerkmale für den Ansatz von Vincent, Liu und Liu 154

11.1 Vergleichsmerkmale für den Ansatz von Vincent et al. 161

12.1 XML Schema Baum für den XML Baum in Abbildung 12.2 164

12.2 XML Baum, entsprechend dem XML Schema Baum in Abbildung 12.1 . 165

12.3 v_{vt}-Subgraphen für das Kontrollbeispiel 5.3 - Teil 1 166

12.4 v_{vt}-Subgraphen für das Kontrollbeispiel 5.3 - Teil 2 166

12.5 v_{vt}-Subgraphen für das Kontrollbeispiel 5.2 - Teil 1 166

12.6 v_{vt}-Subgraphen für das Kontrollbeispiel 5.4 - Teil 2 167

12.7 v_{vt}-Subgraphen für das Kontrollbeispiel 9.4 168

12.8 v_{vt}-Subgraphen für das Kontrollbeispiel 9.5 169

12.9 XML Schema Baum für die XML Bäume in den Abbildungen 4.29 und 4.30 169

12.10XML Schema Baum für die XML Bäume in den Abbildungen 4.29 und 4.30 170

12.11Vergleichsmerkmale für den Ansatz von Hartmann, Link und Kirchberg . 171

13.1 DTD zur Darstellung der Definition von XTFDs 174

13.2 Vergleichsmerkmale für den Ansatz von Mok 179

14.1 Vergleichsmerkmale für den Ansatz von Karlinger 187

15.1 Zusammenfassung aller verglichenen Ansätze 191

Literaturverzeichnis

[1] ABITEBOUL, Serge: Querying Semi-Structured Data. In: *International Conference on Database Theory*, Springer, 1997 (ICDT'07), S. 1–18

[2] ABITEBOUL, Serge ; BUNEMAN, Peter ; SUCIU, Dan: *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kauffman, 1999

[3] ABITEBOUL, Serge ; HULL, Richard ; VIANU, Victor: *Foundations of Databases*. Addison-Wesley Publishing Company, 1995

[4] AHMAD, Kamsuriah ; MAMAT, Ali ; IBRAHIM, Hamidah ; NOAH, Shahrul Azman M.: Defining Functional Dependency for XML. In: *Journal for Information Systems Research and Practice* 1 (2008), Nr. 1

[5] ARENAS, Marcelo ; FAN, Wenfei ; LIBKIN, Leonid: On the Complexity of Verifying Consistency of XML Specifications. In: *SIAM Journal of Computing* 38 (2008), Nr. 3, S. 841–880

[6] ARENAS, Marcelo ; LIBKIN, Leonid: A Normal Form for XML Documents. In: *Symposium on Principles of Database Systems*, 2002, S. 85–96

[7] ARENAS, Marcelo ; LIBKIN, Leonid: A Normal Form for XML Documents. In: *ACM Transactions of Database Systems* 29 (2004), Nr. 1, S. 195–232

[8] ATZENI, Paolo ; DE ANTONELLIS, Valeria: *Relational Database Theory*. Benjamin / Cummings Publishing Company Inc., 1993

[9] BARCELÓ, Pablo ; LIBKIN, Leonid ; REUTTER, Juan L.: On Incomplete XML Documents with Integrity Constraints. In: *AMW* Bd. 619, 2010 (CEUR Workshop Proceedings)

[10] BATINI, Carlo ; SCANNAPIECO, Monica: *Data Quality: Concepts, Methodologies and Techniques*. Springer, 2006 (Data-Centric Systems and Applications)

[11] BENEDIKT, Michael ; FAN, Wenfei ; KUPER, Gabriel M.: Structural Properties of XPath Fragments. In: *Theoretical Computer Science* 336 (2005), S. 3–31

[12] BERGLUND, Anders ; BOAG, Scott ; CHAMBERLIN, Don ; FERNÀNDEZ, Mary F. ; KAY, Michael ; ROBIE, Jonathan ; SIMÉON, Jérôme: XPath Language (XPath) 2.0 / World Wide Web Consortium (W3C). Version: 2007. `http://www.w3.org/TR/xpath20/`. 2007. – Forschungsbericht

[13] BERGLUND, Anders ; FERNÁNDEZ, Mary ; MALHOTRA, Ashok ; MARSH, Jonathan ; NAGY, Marton ; NORMAN, Walsh: XQuery 1.0 and XPath 2.0 Data Model (XDM) / World Wide Web Consortium (W3C). Version: 2010. `http://www.w3.org/TR/xpath-datamodel/`. 2010. – Forschungsbericht

[14] BERTINO, Elisa ; CATANIA, Barbara: Integrating XML and Databases. In: *IEEE Internet Computing* 5 (2001), Nr. 4, S. 84–88

[15] BIRON, Paul V. ; MALHOTRA, Ashok: XML Schema Part 2: Datatypes / World Wide Web Consortium (W3C). Version: Oktober 2004. `http://www.w3.org/TR/xmlschema-2/`. 2004. – Forschungsbericht

[16] BOAG, Scott ; CHAMBERLIN, Don ; FERNÁNDEZ, Mary ; FLORESCU, Daniela ; ROBIE, Jonathan ; SIMÉON, Jérôme: XQuery 1.0: An XML Query Language / World Wide Web Consortium (W3C). Version: 2007. `http://www.w3.org/TR/xquery/`. 2007. – Forschungsbericht

[17] BRAY, Tim ; PAOLI, Jean ; SPERBERG-MCQUEEN, Michael ; MALER, Eve ; YERGEAU, Francois: Extensible Markup Language (XML) 1.0. / World Wide Web Consortium (W3C). Version: November 2008. `http://www.w3.org/TR/xml/`. 2008. – Forschungsbericht

[18] BUNEMAN, Peter ; DAVIDSON, Susan ; FAN, Wenfei ; HARA, Carmem ; TAN, Wang chiew: Reasoning about Keys for XML. In: *Database Programming Languages* Bd. 2397, Springer, 2001 (Lecture Notes in Computer Science), S. 133–148

[19] BUNEMAN, Peter ; DAVIDSON, Susan ; FAN, Wenfei ; HARA, Carmem ; TAN, Wang-Chiew: Reasoning about Keys for XML. In: *Information Systems* 28 (2003), Nr. 8, S. 1037–1063

[20] BUNEMAN, Peter ; DAVIDSON, Susan B. ; FAN, Wenfei ; HARA, Carmem S. ; TAN, Wang-Chiew: Keys for XML. In: *Computer Networks* 39 (2002), Nr. 5, S. 473–487

[21] BUNEMAN, Peter ; FAN, Wenfei ; SIMÉON, Jérôme ; WEINSTEIN, Scott: Constraints for Semistructured Data and XML. In: *ACM SIGMOD Record* 30 (2001), Nr. 1, S. 47–54

[22] BUNEMAN, Peter ; FAN, Wenfei ; WEINSTEIN, Scott: Interaction between Path and Type Constraints. In: *ACM Transactions on Computational Logics* 4 (2003), Nr. 4, S. 530–577

[23] CHEN, Haitao ; LIAO, Husheng ; GAO, Zenggi: Functional Dependencies for XML. In: *WAIM Workshops* Bd. 6185, Springer, 2010, S. 110–115

[24] CHEN, Haitao ; LIAO, Husheng ; GAO, Zenggi: On Defininig Functional Dependencies in XML Schema. In: *FGIT-DTA/BSBT* Bd. 118, Springer, 2010 (Communications in Computer and Information Science), S. 120–131

[25] CLARK, James ; MURATA, Makoto: RELAX NG Specification / The Organization for the Advancement of Structured Information Standards (OASIS). Version: Dezember 2001. `https://www.oasis-open.org/committees/relax-ng/spec-20011203.html`. 2001. – Forschungsbericht

[26] COWAN, John ; TOBIN, Richard: XML Information Set (Second Edition) / World Wide Web Consortium (W3C). Version: Februar 2004. `http://www.w3.org/TR/2004/REC-xml-infoset-20040204/`. 2004. – Forschungsbericht

[27] DAVIDSON, Andrew ; FUCHS, Matthew ; HEDIN, Mette ; JAIN, Mudita ; KOISTINEN, Jari ; LLYD, Chris ; MALONEY, Murray ; SCHWARZHOF, Kelly: Schema for Object-Oriented XML 2.0 / World Wide Web Consortium (W3C). Version: Juli 1999. `http://www.w3.org/TR/NOTE-SOX`. 1999. – Forschungsbericht

[28] DEROSE, Steve ; MALER, Eve ; ORCHARD, David ; NORMAN, Walsh: XML Linking Language (XLink) Version 1.1 / World Wide Web Consortium (W3C). Version: Mai 2010. `http://www.w3.org/TR/xlink11/`. 2010. – Forschungsbericht

[29] DIESTEL, Reinhard: *Graph Theory*. 3. Springer, 2005 (Graduate Texts in Mathematics 173)

[30] ELMASRI, Ramez ; NAVATHE, Shamkant: *Fundamentals of Database Systems*. Bd. 6. Addison-Wesley Publishing Company, 2010

[31] FAGIN, Ronald ; VARDI, Moshe: The Theory of Data Dependencies - An Overview. In: *Mathematics of Information Processing* Bd. 34 American Mathematical Society, Springer, 1986 (Proceedings of Symposia in Applied Mathematics), S. 19–71

[32] FALLSIDE, David C. ; WALMSLEY, Priscilla: XML Schema Part 0: Primer / World Wide Web Consortium (W3C). Version: October 2004. `http://www.w3.org/TR/xmlschema-0/`. 2004. – Forschungsbericht

[33] FAN, Wenfei: XML Constraints: Specification, Analysis, and Applications. In: *Database and Expert Systems Applications Workshops*, IEEE Computer Society, 2005, S. 805–809

[34] FAN, Wenfei ; LIBKIN, Leonid: On XML Integrity Constraints in the Presence of DTDs. In: *ACM* 49 (2002), Mai, Nr. 3, S. 368–406

[35] FAN, Wenfei ; SIMÉON, Jérôme: Integrity Constraints for XML. In: *Computer and System Sciences* 66 (2003), Nr. 1, S. 254–291. – ISSN 0022–0000

[36] FERRAROTTI, Flavio ; HARTMANN, Sven ; LINK, Sebastian ; WANG, Jing: Promoting the Semantic Capability of XML Keys. In: *XSym'10* Bd. 6309, Springer, 2010 (Lecture Notes in Computer Science), S. 144–153

[37] GROSSO, Paul ; MALER, Eve ; MARSH, Jonathan ; WALSH, Norman: XPointer Framework / World Wide Web Consortium (W3C). Version: März 2003. `http://www.w3.org/TR/xptr-framework/`. 2003. – Forschungsbericht

[38] HARTMANN, Sven ; KÖHLER, Henning ; LINK, Sebastian ; TRINH, Thu ; WANG, Jing: On the Notion of an XML Key. In: *Semantics in Data and Knowledge Bases* Bd. 4925, Springer, 2008 (Lecture Notes in Computer Science), S. 103–112

[39] HARTMANN, Sven ; LINK, Sebastian: More Functional Dependenies for XML. In: *Advances in Databases and Information Systems* Bd. 2798, Springer, 2003 (Lecture Notes in Computer Science), S. 355–369

[40] HARTMANN, Sven ; LINK, Sebastian: Multi-valued Dependencies in the Presence of Lists. In: *PODS*, ACM, 2004, S. 330 – 341

[41] HARTMANN, Sven ; LINK, Sebastian: Efficient Reasoning about a Robust XML Key Fragment. In: *ACM Trasactions on Database Systems* 34 (2009), Nr. 2

[42] HARTMANN, Sven ; LINK, Sebastian: Expressive, yet Tractable XML Keys. In: *International Conference on Extending Database Technology* Bd. 360, ACM, 2009 (International Conference Proceeding Series), S. 357–367

[43] HARTMANN, Sven ; LINK, Sebastian ; KIRCHBERG, Markus: A Subgraph-based Approach Towards Functional Dependencies for XML. In: *World-Multiconference on Systemics, Sybernetics and Informatics* Bd. 9, 2003 (Computer Science and Engineering II), S. 200–205

[44] HARTMANN, Sven ; LINK, Sebastian ; SCHEWE, Klaus-Dieter: Reasoning about Functional and Multi-valued Dependencies in the Presence of Lists. In: *Foundations of Information and Knowledge Systems (FoIKS)*, Springer, 2004 (Lecture Notes in Computer Science), S. 134–154

[45] HARTMANN, Sven ; TRINH, Thu: Axiomatising Functional Dependencies for XML with Frequencies. In: *International Symposium on Founcations of Information and Knowledge Systems*, 2006, S. 159–178

[46] HEINRICH, Lutz J. ; HEINZL, Armin ; ROITHMAYR, Friedrich: *Wirtschaftsinformatik-Lexikon*. 7. Oldenbourg, 2004

[47] JAN BEX, Geert ; NEVEN, Frank ; BUSSCHE, Jan Van d.: DTDs versus XML Schema: A Practical Study. In: *WebDB*, 2004, S. 79–84

[48] JELLIFFE, Rick: Schematron / ISO/IEC. Version: 2006. `http://www.ascc.net/xml/resource/schematron/`. 2006 (19757-3). – Forschungsbericht

[49] KARLINGER, Michael: *Keys and Foreign Keys fo XML: Design and Reasoning*, Johannes Kepler Universität Linz, Institut für Wirtschaftsinformatik - Data & Knowledge Engineering, Diss., 2010

[50] KARLINGER, Michael ; VINCENT, Millist W. ; SCHREFL, Michael: Inclusion Dependencies in XML: Extending Relational Semantics. In: *Database and Expert Systems Applications* Bd. 5690, Springer, 2009 (Lecture Notes in Computer Science), S. 23–37

[51] KARLINGER, Michael ; VINCENT, Millist W. ; SCHREFL, Michael: Keys in XML: Capturing Identification and Uniqueness. In: *Web Information Systems Engineering* Bd. 5802, Springer, 2009 (Lecture Notes in Computer Science), S. 563–571

[52] KAZAKOS, Wassili ; SCHMIDT, Andreas ; TOMCZYK, Peter: *Datenbanken und XML: Konzepte, Anwendungen, Systeme*. Springer, 2002

[53] KLARLUND, Nils ; SCHWENTICK, Thomas ; SUCIU, Dan: XML: Model, Schemas, Types, Logics and Queries. In: *Logics for Emerging Applications of Databases*, Springer, 2003, S. 1–41

[54] KLETTKE, Meike ; MEYER, Holger: *XML und Datenbanken*. dpunkt-Verlag, 2003

[55] KOT, Lucja ; WHITE, Walker: Characterization of the Interaction of XML Functional Dependencies with DTDs. In: *International Conference on Database Theory* Bd. 4353, Springer, 2007 (Lecture Notes in Computer Science), S. 119–133

[56] LE HORS, Arnaud ; LE HÉGARET, Phillippe ; WOOD, Lauren ; NICOL, Gabin ; ROBIE, Jonathan ; CHAMPION, Mike ; BYREN, Steve: Document Object Model (DOM) Level 3 Core Sprecification / World Wide Web Consortium (W3C). Version: 2004. `http://www.w3.org/TR/DOM-Level-3-Core/`. 2004. – Forschungsbericht

[57] LEE, Dongwon ; CHU, Wesely W.: Comparative Analysis of Six XML Schema Languages. In: *ACM SIGMOD Record* 29 (2000), Nr. 3, S. 76–87

[58] LEE, Mong-Li ; LING, Tok W. ; LOW, Wai L.: Designing Functional Dependencies for XML. In: *International Conference on Extending Database Technology: Advances in Database Technology* Bd. 2287, Springer, 2002 (Lecture Notes in Computer Science), S. 124–141

[59] LIU, Jixue ; VINCENT, Millist W. ; LIU, Chengfei: Local XML Functional Dependencies. In: *International Workshop on Web Information and Data Management*, ACM, 2003, S. 23–28

[60] LV, Teng ; YAN, Ping: XML Constraint-tree-based Functional Dependenies. In: *International Conference on e-Business Engineering* IEEE Computer Society, 2006, S. 224–228

[61] LV, Teng ; YAN, Ping: A Survey Study on XML Functional Dependencies. In: *International Symposium on Data, Privacy, and E-Commerce*, 2007, S. 143–145

[62] MAIER, David: *The Theory of Relational Database*. Computer Science Press, 1983

[63] MILOSLAV, Nic: *Schematron Tutorial*. `http://www.zvon.org/HTMLonly/SchematronTutorial/General/contents.htm`. Version: Mai 2000

[64] MOK, Wai Y.: On Utilizing Variables for Specifying FDs in Data-centric XML Documents. In: *Data & Knowledge Engineering* 60 (2007), Nr. 3, S. 494–510

[65] MÖLLER, Anders: *Document Structure Description 2.0*. `http://www.brics.dk/DSD/dsd2.html`. Version: Dezember 2002

[66] MÖLLER, Anders ; SCHWARTZBACH, Michael I.: *An Introduction to XML and Web Technologies*. Addison-Wesley Publishing Company, 2006

[67] NG, Wilfred: An Extension of the Relational Data Model to Incorporate Ordered Domains. In: *ACM Transactions of Database Systems* 26 (2001), Nr. 3, S. 344–383

[68] PEMBERTON, Steven et a.: XHTML 1.0: The Extensible HyperText Markup Language / World Wide Web Consortium (W3C). Version: Jänner 2000. `http://www.w3.org/TR/xhtml1/`. 2000. – W3C Recommendation

[69] RAGGETT, Dave ; LE HORS, Arnaud ; JACOBS, Ian: HTML 4.01 Specification / World Wide Web Consortium (W3C). Version: Dezember 1999. `http://www.w3.org/TR/html401`. 1999. – W3C Recommendation

[70] RAMAKRISHNAN, Raghu ; GEHRKE, Johannes: *Database Management Systems*. McGraw-Hill Higher Education, 2000

[71] SAHUGUET, Arnaud: Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask. In: *International Workshop WebDB* Bd. 1997, Springer, 2000 (Lecture Notes in Computer Science), S. 171–183

[72] SALL, Kenneth B.: *XML Family of Specifications*. Addison-Wesley Publishing Company, 2002 `http://wdvl.internet.com/Authoring/Languages/XML/XMLFamily/XMLSyntax/`

[73] SHAHRIAR, Md. S. ; LIU, Jixue: On the Performances of Checking XML Key and Functional Dependency Satisfactions. In: *On the Move to Meaningful Internet Systems Conferences* Bd. 5871, Springer, 2009 (Lecture Notes in Computer Science), S. 1254–1271

[74] SHAHRIAR, Sumon ; LIU, Jixue: On Defining Keys for XML. In: *International Conference on Computer and Information Technology Workshops*, IEEE Computer Society, 2008, S. 86–91

[75] SHAHRIAR, Sumon ; LIU, Jixue: On Defining Functional Dependency for XML. In: *International Conference on Semantic Computing*, IEEE Computer Society, 2009, S. 595–600

[76] THOMPSON, Henry S. ; BEECH, David ; MALONEY, Murray ; MENDELSOHN, Noah: XML Schema Part 1: Structures / World Wide Web Consortium (W3C). Version: Oktober 2004. `http://www.w3.org/TR/xmlschema-1/`. 2004. – Forschungsbericht

[77] THURAISINGHAM, Bhavani: *XML Databases and the Semantic Web*. CRC Press, 2002

[78] VIANU, Victor: A Web Odyssey: from Codd to XML. In: *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, ACM Press, Juni 2001, S. 1–15

[79] VINCENT, Millist W. ; LIU, Jixue: Functional Dependencies for XML. In: *Fith Asian Pacific Web Conference* Bd. 2642, Springer, 2003 (Lecture Notes in Computer Science), S. 22–34

[80] VINCENT, Millist W. ; LIU, Jixue: Multivalued Dependencies and a 4NF for XML. In: *International Conference on Advanced Information Systems Engineering*, Springer, 2003 (Lecture Notes in Computer Science), S. 14–29

[81] VINCENT, Millist W. ; LIU, Jixue: Multivalued Dependencies in XML. In: *British National Conference on Databases* Bd. 2712, Springer, 2003 (Lecture Notes in Computer Science), S. 4–18

[82] VINCENT, Millist W. ; LIU, Jixue: Checking Functional Dependency Satisfaction in XML. In: *International XML Database Symposium* Bd. 3671, Springer, 2005 (Lecture Notes in Computer Science), S. 4–17

[83] VINCENT, Millist W. ; LIU, Jixue ; LIU, Chengfei: Strong Functional Dependencies and a Redundancy Free Normal Form for XML. In: *World Multi-Conference on Systemics, Cybernetics and Infromatics*, 2003

[84] VINCENT, Millist W. ; LIU, Jixue ; LIU, Chengfei: Strong Functional Dependenies and Their Application to Normal Forms in XML. In: *ACM Transactions on Database Systems* 29 (2004), September, Nr. 3, S. 445–462

[85] VINCENT, Millist W. ; LIU, Jixue ; MOHANIA, Mukesh: On the Equivalence between FDs in XML and FDs in Relations. In: *Acta Informatica* 44 (2007), Nr. 3, S. 207–247

[86] VINCENT, Millist W. ; SCHREFL, Michael ; LIU, Jixue ; LIU, Chengfei ; DOGEN, Solen: Generalized Inclusion Dependenies in XML. In: *Asia-Pasific Web Conference* Bd. 3007, Springer, 2004 (Lecture Notes in Computer Science), S. 224–233

[87] WANG, Junhu: A Comparative Study of Functional Dependencies for XML. In: *Asia-Pacific Web Conference* Bd. 3399, Springer, 2005 (Lecture Notes in Computer Science), S. 308–319

[88] WANG, Junhu ; TOPOR, Rodney: Removing XML Data Redundancies Using Functional and Equality-Generating Dependencies. In: *Australasian Database Conference* Bd. 39, Australian Computer Society, 2005 (CRPIT), S. 65–74