

Diplomarbeit

Erweiterung und Optimierung der GCX Engine
– Pfade mit mehreren Schritten und Aggregatfunktionen –

Gunnar Jehl



Albert-Ludwigs-Universität Freiburg

Fakultät für Angewandte Wissenschaften

Institut für Informatik

Lehrstuhl für Datenbanken und Informationssysteme

Prof. Dr. Georg Lausen

Oktober 2008

Albert-Ludwigs-Universität Freiburg
Fakultät für Angewandte Wissenschaften
Institut für Informatik

Lehrstuhl für Datenbanken und Informationssysteme
Prof. Dr. Georg Lausen



Diplomarbeit

Erweiterung und Optimierung der GCX Engine

– Pfade mit mehreren Schritten und Aggregatfunktionen –

Gunnar Jehl

Bearbeiter: Gunnar Jehl
Matrikelnummer: 1325893

Gutachter: Prof. Dr. Georg Lausen
Datenbanken und Informationssysteme
Prof. Dr. Andreas Podelski
Softwaretechnik

Betreuer: Prof. Dr. Georg Lausen
Dipl.-Inf. Michael Schmidt

Abgabedatum: 29. Oktober 2008

Gunnar Jehl
Hochdorferstr. 17
79108 Freiburg-Hochdorf
E-Mail: jehl@informatik.uni-freiburg.de

Diese Arbeit wurde mit Hilfe von KOMA-Script und pdfL^AT_EX gesetzt.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst habe, keine anderen als die angegebenen Quellen/Hilfsmittel verwendet habe und alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Schriften entnommen wurden, als solche kenntlich gemacht habe. Darüber hinaus erkläre ich, dass diese Arbeit nicht, auch nicht auszugsweise, bereits für eine andere Prüfung angefertigt wurde.

Freiburg, den 29.10.2008

Gunnar Jehl

Danksagung

Die vorliegende Diplomarbeit wurde am Lehrstuhl für Datenbanken und Informationssysteme der Albert-Ludwigs-Universität Freiburg unter der Leitung von Prof. Dr. Georg Lausen angefertigt.

An erster Stelle bedanke ich mich bei *Prof. Dr. Georg Lausen* für die Überlassung des hoch spannenden Themas und die hilfreiche Unterstützung, die mir eine angemessene Ausarbeitung der Aufgabenstellung ermöglicht hat.

Ein ganz besonderer Dank gilt meinem Betreuer *Dipl.-Inf. Michael Schmidt* für seine großartige, engagierte und lehrreiche Betreuung, für seine Ratschläge und Hilfestellungen bei den unterschiedlichen Problemen und der Suche nach Lösungswegen im Rahmen der Anfertigung meiner Diplomarbeit. *Michaels* hervorragende Kenntnis auf diesem Gebiet, seine Unterstützung, sein Verständnis und auch seine Geduld haben wesentlich zum Gelingen und erfolgreichen Abschluss dieser Arbeit beigetragen. Danke *Michael* für die vielen fachlichen Diskussionen, konstruktiven Überlegungen, Gespräche und E-Mails, ohne beim „BufferIterator“ den Verstand zu verlieren.

Ebenso möchte ich mich bei *Prof. Dr. Andreas Podelski* für die freundliche Übernahme des Zweitgutachtens bedanken.

Weiter danke ich meinem Studienkollegen *Alexander Kortus*, der die Fortschritte meiner Diplomarbeit prüfte, ein mitfühlendes Ohr hatte und ein inspirierendes Beispiel bot. Ein herzliches Dankeschön geht auch an *Thomas Fisch* und *Sabine Schwarz* für die Gespräche und ihr Zuhören. Sie waren es, die mit dazu beitrugen, die Probleme kleiner erscheinen zu lassen als zuvor.

Zuletzt und in besonderem Maße gilt mein Dank meiner Mutter *Claudia Jehl*, die mich während meines gesamten Studiums und der Diplomarbeitsphase mit Anteilnahme, hilfreichen Anregungen und Anmerkungen und schließlich durch ihr geduldiges Korrekturlesen dieser Arbeit eindrucksvoll unterstützt hat. Sie, meine Großmutter *Hedwig Metzger* und mein Patenonkel *Joachim Metzger* waren es, die mir einen starken familiären Rückhalt und eine einzigartige Unterstützung boten und dadurch überhaupt erst mein Studium und diese Arbeit möglich gemacht haben.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Ziele	3
1.3	Aufbau	4
2	Extensible Markup Language (XML)	7
2.1	Bestandteile und Namenskonventionen	7
2.2	Darstellungsformen	8
2.3	XML Path Language (XPath)	11
2.4	XML Query Language (XQuery)	15
3	Aktive Speicherbereinigung: Garbage Collected XQuery (GCX)	19
3.1	Sprachfragment XQ	20
3.2	Statische Analyse	24
3.2.1	Funktion und Herleitung eines Projektionsbaums	24
3.2.2	Aufgabe und Einfügen von SignOff-Anweisungen	32
3.3	Dynamische Analyse	39
3.3.1	Zuweisen und Entfernen von Rollen	40
3.3.2	Ausführen der Speicherbereinigung	42
3.4	Praktische Umsetzung: Garbage Collected XQuery (GCX)	43
4	GCX Erweiterung: Pfade mit mehreren Schritten	53
4.1	Pfade mit einem Schritt vs. Pfade mit mehreren Schritten	53
4.2	Sprachfragment XQ'	55
4.3	Theoretische Grundlagen und Hintergründe	57
4.4	Implementierungen	65
5	GCX Erweiterung: Aggregatfunktionen	69
5.1	Sprachfragment XQ''	69
5.2	Idee zur Einbettung der Aggregatfunktionen	74
5.3	Theoretische Grundlagen und Hintergründe	77
6	GCX Optimierungen	81
6.1	Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden	81

6.2	Entfernen von Tupeln in Abhängigkeiten bei enthaltenen Knotenmengen	84
6.3	Entfernen von nicht benötigten Knoten eines Projektionsbaums	89
6.4	Entfernen von redundanten Rollen eines Projektionsbaums	92
6.5	Fixierung der Auswertungsreihenfolge von SignOff-Anweisungen	99
6.6	Gegensätzliche Optimierungen	105
7	Experimentelle Resultate	109
7.1	Grundlagen und Aufbau	110
7.2	Experimentelle Ergebnisse	114
7.3	Diskussion der Ergebnisse	119
8	Zusammenfassung und Ausblick	125
	Abbildungsverzeichnis	129
	Tabellenverzeichnis	131
	Quelltextverzeichnis	133
	Abkürzungsverzeichnis	135
	Literaturverzeichnis	137
A	Übersicht der beiliegenden CD	145
A.1	Ordnerstruktur und Ordnerbeschreibungen	145
B	Extensible Markup Language (XML)	147
B.1	Beispiel XML-Dokument	147
B.2	Beispiele XPath-Achsen	148
C	Aktive Speicherbereinigung: Garbage Collected XQuery (GCX)	155
C.1	Beispiel der aktiven Speicherbereinigung	155
D	GCX Erweiterung: Pfade mit mehreren Schritten	173
D.1	XML-Dokument	173
E	GCX Erweiterung: Aggregatfunktionen	175
E.1	XML-Dokument	175
F	GCX Optimierungen	177
F.1	XML-Dokument	177

G	Experimentelle Resultate	179
G.1	XMark Testanfragen	179
G.1.1	Pfade mit einem Schritt	179
G.1.1.1	XMark-Anfrage Q01	179
G.1.1.2	XMark-Anfrage Q02	179
G.1.1.3	XMark-Anfrage Q08	180
G.1.1.4	XMark-Anfrage Q09	180
G.1.1.5	XMark-Anfrage Q11	181
G.1.1.6	XMark-Anfrage Q12	181
G.1.1.7	XMark-Anfrage Q13	182
G.1.1.8	XMark-Anfrage Q14	182
G.1.1.9	XMark-Anfrage Q15	182
G.1.1.10	XMark-Anfrage Q18	183
G.1.1.11	XMark-Anfrage Q19	183
G.1.2	Pfade mit mehreren Schritten	184
G.1.2.1	XMark-Anfrage Q01	184
G.1.2.2	XMark-Anfrage Q02	184
G.1.2.3	XMark-Anfrage Q03	184
G.1.2.4	XMark-Anfrage Q05	185
G.1.2.5	XMark-Anfrage Q06	185
G.1.2.6	XMark-Anfrage Q07	185
G.1.2.7	XMark-Anfrage Q08	185
G.1.2.8	XMark-Anfrage Q09	186
G.1.2.9	XMark-Anfrage Q10	186
G.1.2.10	XMark-Anfrage Q11	187
G.1.2.11	XMark-Anfrage Q12	187
G.1.2.12	XMark-Anfrage Q13	187
G.1.2.13	XMark-Anfrage Q14	188
G.1.2.14	XMark-Anfrage Q15	188
G.1.2.15	XMark-Anfrage Q16	188
G.1.2.16	XMark-Anfrage Q17	188
G.1.2.17	XMark-Anfrage Q18	189
G.1.2.18	XMark-Anfrage Q19	189
G.1.2.19	XMark-Anfrage Q20	189

Kapitel 1

Einleitung

1.1 Motivation

Mit dem stetig wachsenden World Wide Web (WWW) ist auch das Interesse nach Anwendungen und Anwendungsgebieten zur Darstellung, Verarbeitung und Speicherung von Daten und zum elektronischen Datenaustausch gestiegen.

Die Verarbeitung und der Austausch von Informationen über das Internet wird schon jetzt vielfach in biologischen und astronomischen Bereichen und in der Meteorologie, aber auch in der Tourismussparte und im Bibliothekssektor, genutzt.

Durch die Zunahme von Netzwerk-Anwendungen, der raschen Verbreitung des Internets und den damit einhergehenden steigenden Datenübertragungsraten ist es erforderlich, ein einheitliches Format für den Datenaustausch zu verwenden und die vielseitigen Datenströme zu vereinen und die Daten verschiedener Systeme untereinander auszutauschen. Dabei haben sich neue Anforderungen zur Nutzung der übertragenen Daten und zur Verarbeitung von Datenströmen ergeben.

Vor diesem Hintergrund entstand die *Extensible Markup Language (XML)* als Standard des World Wide Web Consortium (W3C). XML bezeichnet dabei ein Dokumentenformat und dient zur Darstellung (semi-)strukturierten Daten, deren Speicherung und elektronischen Austauschs. In den letzten Jahren entwickelte sich XML dabei zu einem Standardformat für den Datenaustausch [20].

Wegen der großen Bedeutung und Verwendung von XML in Form von XML-Dokumenten wurden in den letzten Jahren verschiedene Anfragesprachen mit unterschiedlicher Fokussierung für den Zugriff auf XML-Daten entwickelt.

Als wesentliche Anfragesprache hat sich dabei *XML Query Language (XQuery)* herauskristallisiert. XQuery ist eine vom W3C spezifizierte Anfragesprache für

XML(-Dokumente). Sie stellt dabei nach [16] eine sehr ausdrucksmächtige, funktionale Sprache dar, deren Basis Ausdrücke sind. Der Grundstein von XQuery bildet *XML Path Language (XPath)*. XPath ist nach [18] keine vollständige Anfragesprache, sondern dient der Adressierung und Auswahl von Bestandteilen eines XML-Dokuments.

Die Auswertung von XQuery-Anfragen auf ein als Datenstrom erhaltenes XML-Dokument ist aufgrund großer Datenmengen und hohen Übertragungsraten problematisch. Durch die schnelle Übertragung der riesig bis sogar unendlich großen Datenmengen ist ein Zwischenspeichern der erhaltenen XML-Daten und deren Verarbeitung im – in der Praxis meist stark limitiertem – Hauptspeicher nicht immer möglich [20].

Aus diesem Grund ist es sinnvoll und auch notwendig, die erhaltenen XML-Dokumente schon zu Beginn zu „filtern“. Dazu kann die in [1, 29, 31] beschriebene *Dokumentprojektion* genutzt werden, die eine etablierte Technik zur Minimierung des Speicherbedarfs darstellt. Diese weit verbreitet eingesetzte Technik allein reicht allerdings nicht aus, weshalb es zusätzlich einer effizienten Speicherverwaltung bedarf, mit deren Hilfe XML-Daten frühzeitig wieder aus dem Speicher entfernt werden können.

An diesem Punkt setzt die in [20] bzw. [21] vorgestellte *aktive Speicherbereinigung* an. Sie ist eine Kombination aus *statischer* und *dynamischer Analyse* bei der Auswertung von XQuery-Anfragen auf ein als Datenstrom erhaltenes XML-Dokument mit dem Ziel der Reduzierung des Speicherbedarfs. Die statische Analyse erfolgt dabei vor der Auswertung einer XQuery-Anfrage bzw. vor der Verarbeitung des als Datenstrom erhaltenen XML-Dokuments. Durch sie werden die Grundsteine für die dynamische Analyse gelegt. Genauer gesagt wird durch sie der für eine XQuery-Anfrage zugehörige Projektionsbaum erstellt, der die zur Auswertung einer XQuery-Anfrage relevanten XML-Daten bestimmt. Weiter werden von ihr SignOff-Anweisungen in eine XQuery-Anfrage eingefügt. SignOff-Anweisungen besitzen die Aufgabe, XML-Daten, die ihre Relevanz für die Auswertung einer XQuery-Anfrage verloren haben, mit Hilfe der dynamischen Analyse aus dem Speicher zu entfernen. Den Indikator für die Relevanz von gespeicherten XML-Daten stellen dabei so genannte Rollen dar. Diese werden den XML-Daten, die gespeichert werden, zugewiesen. Die dynamische Analyse erfolgt während der Auswertung ei-

ner XQuery-Anfrage und während der Verarbeitung des als Datenstrom erhaltenen XML-Dokuments und vollzieht die Zuweisung von Rollen den zu speichernden XML-Daten. Dieses Zuweisen ähnelt dem Prinzip des Zählens von Referenzen (reference counting), eine weit verbreitete Technik in Programmiersprachen zur automatischen Speicherbereinigung. Bei dieser Technik erhält jedes Objekt einen Zähler, der erhöht wird, falls eine Referenz hinzukommt. Analog wird dieser beim Entfernen einer Referenz verringert. Falls der Zähler eines Objekts null wird, so kann dieses Objekt, da kein Zugriff auf dieses mehr möglich ist, gelöscht werden, um den von diesem Objekt belegten Speicher wieder freizugeben. Im Gegensatz zur meist passiv angestoßenen Speicherbereinigung, d.h. sobald kein Speicher mehr zu Verfügung steht, erfolgt diese bei der aktiven Speicherbereinigung – wie der Name schon sagt – aktiv. Das aktive Anstoßen der Speicherbereinigung erfolgt dabei durch die von der statischen Analyse eingefügten SignOff-Anweisungen, die durch ihre Ausführung die Rollen sukzessive von den gespeicherten XML-Daten entfernen. Falls XML-Daten keine Rolle mehr besitzen, vergleichbar mit einem Zählerstand von null beim Zählen von Referenzen (reference counting), werden sie aus dem Speicher entfernt.

An dieser Stelle setzt diese Arbeit an.

1.2 Ziele

Ausgangspunkt ist die aktive Speicherbereinigung und deren praktische Umsetzung in der *Garbage Collected XQuery (GCX)* Engine mit einem existierenden Framework zur Auswertung von XQuery-Anfragen auf XML-Dokumente, die als Datenstrom erhalten werden.

Diese beiden sollten um die Fähigkeit erweitert werden, XQuery-Anfragen zu unterstützen, die XPath-Lokationspfade mit mehreren XPath-Lokationsschritten besitzen.

Zusätzlich sollten Aggregatfunktionen in das bestehende Framework der aktiven Speicherbereinigung bzw. der GCX Engine umgesetzt werden. Hierzu sollten die Aggregatfunktionen nicht nur schlicht in das vorhandene Framework implementiert, sondern vielmehr darin eingebettet werden.

Bei der Frage, wie sich die zur Auswertung einer XQuery-Anfrage benötigte Zeit und der dazu notwendige Speicherbedarf (weiter) reduzieren lassen, rücken Optimierungen unweigerlich in den Fokus. Im Rahmen dieser Arbeit sollten daher weitere Optimierungsmöglichkeiten ausfindig gemacht werden, die die Auswertungszeit und den Speicherbedarf (weiter) reduzieren.

Um den Wert der aktiven Speicherbereinigung bzw. der GCX Engine durch die im Rahmen dieser Arbeit hinzugefügten Erweiterungen und Optimierungen hinsichtlich der zur Auswertung einer XQuery-Anfrage benötigten Zeit und den dazu notwendigen Speicherbedarf einschätzen zu können, sollten Versuchsreihen durchgeführt werden und die GCX Engine mit anderen existierenden XQuery Engines verglichen werden.

1.3 Aufbau

Diese Arbeit besteht aus acht Kapiteln. Nach diesem ersten Kapitel, das einleitend die Motivation und Ziele beschreibt, werden in Kapitel 2 die für das Verständnis dieser Arbeit notwendigen Grundlagen der *Extensible Markup Language (XML)*, *XML Path Language (XPath)* und *XML Query Language (XQuery)* erläutert.

In Kapitel 3 wird auf die in [20] bzw. [21] vorgestellte *aktive Speicherbereinigung* und deren praktische Umsetzung in der *Garbage Collected XQuery (GCX) Engine* eingegangen.

Aufbauend auf diesen Ausgangspunkten wird in Kapitel 4 die Erweiterung zur Unterstützung von XQuery-Anfragen, die XPath-Lokationspfade mit mehreren XPath-Lokationsschritten besitzen, präsentiert.

Kapitel 5 beschreibt die Einbettung der Aggregatfunktionen in das vorhandene Framework der aktiven Speicherbereinigung bzw. der GCX Engine.

In Kapitel 6 werden insgesamt fünf Optimierungen vorgestellt, die noch während der statischen Analyse erfolgen und das Ziel haben, die Auswertungszeit zu beschleunigen und/oder den Speicherbedarf zu reduzieren.

Kapitel 7 beschreibt zunächst die Grundlagen und den Aufbau der durchgeführten Versuchsreihen. Anschließend werden tabellarisch die einzelnen Ergebnisse präsentiert und zuletzt die Ergebnisse analysiert, verglichen und diskutiert.

Den Schluss bildet Kapitel 8, das die wesentlichen Teile und die erzielten Ergebnisse dieser Arbeit zusammenfasst und einen Ausblick auf weitere Möglichkeiten zur Erweiterung und Optimierung der aktiven Speicherbereinigung bzw. der GCX Engine gibt.

Kapitel 2

Extensible Markup Language (XML)

Dieses Kapitel bildet die Grundlage der gesamten weiteren Arbeit. In ihm werden zunächst die Bestandteile und Namenskonventionen der *Extensible Markup Language (XML)* von [32] eingeführt. Darauf aufbauend werden die Darstellungsmöglichkeiten von XML-Dokumenten präsentiert und die Kernpunkte von *XML Path Language (XPath)* von [33, 34] sowie *XML Query Language (XQuery)* von [36] beschrieben. Dabei ist im Wesentlichen für XML auf [10], für XPath auf [11] und für XQuery auf [12, 13] zurückgegriffen und Teile im Ganzen oder teilweise übernommen worden.

2.1 Bestandteile und Namenskonventionen

XML-Dokumente (kurz: Dokument/-e) bestehen, wie durch Quelltext B.1 verdeutlicht, aus *Tags* (`<tag>` öffnender Tag, `</tag>` schließender Tag und `<tag/>` dem leeren Element(-Tag)). Diese bilden die *Elemente* (öffnender und sein schließender Tag zusammen *mit* dem von ihnen eingeschlossenen Ausschnitt des Dokuments oder dem leeren Element bei leerem Element(-Tag)) mit *Inhalten der Elemente* (eingeschlossener Ausschnitt des Dokuments *ohne* öffnenden und schließenden Tag oder dem leerem Inhalt bei leerem Element(-Tag)). Dabei können evtl. *Attribute* (`</tag attr1 = "val1" ... attrn = "valn">` mit $n \geq 1$; Zusatzinformationen eines Elements) innerhalb der öffnenden Tags der Elemente stehen. Die u.U. weiter vorhandenen Bestandteile eines Dokuments, wie *XML-Deklaration*, *Kommentare*, *Dokumenttypdefinition (Document Type Definition (DTD))*, *Verarbeitungsanweisungen (processing instructions)*, *Entitätenreferenzen* und *character data (CDATA)*-Abschnitte, sind innerhalb dieser Arbeit ohne Bedeutung und werden daher außer Acht gelassen.

Der *Elementname* ist der Name seines bzw. seiner zueinander gehörenden Tags. Der *Inhalt eines Elements* kann entweder aus *Elementtext* – auch *parsed character data* (*PCDATA*) genannt –, also einer alphanumerischen Zeichenkette¹ (ohne Tags), aus *Elementinhalt*, also aus weiteren Elementen, oder aus *gemischtem Inhalt*, also aus einer Mischung von beiden, bestehen.

Eine Forderung in Dokumenten ist deren *Wohlgeformtheit* (*well-formedness*). Einige Kriterien, die hierfür nach [32] erfüllt sein müssen, verlangen, dass jedes/in jedem Dokument (1) mindestens ein oder mehrere Elemente enthält, (2) ein Element, das *Wurzel-* oder *Dokumentelement* genannt wird, das in keinem anderen Element enthalten ist und keinen Inhalt eines anderen Elements darstellt, besitzt, (3) die Namen der zueinander gehörenden öffnenden und schließenden Tags übereinstimmen und (4) Elemente mit Inhalt korrekt verschachtelt sind, d.h. falls der öffnende Tag eines Elements E den Inhalt eines anderen Elements E' darstellt, muss sich der schließende Tag von E ebenfalls im Inhalt von E' befinden.

2.2 Darstellungsformen

Dokumente können im Allgemeinen aus zwei unterschiedlichen Sichtweisen, zwischen denen in der weiteren Arbeit ständig gewechselt wird, betrachtet werden. So können diese entweder – wie in Datenströmen – als Zeichenkette, d.h. aus Elementen mit Attributen in den öffnenden Tags dieser, oder durch einen *XML-Dokumentenbaum* (kurz: Dokumentenbaum) dargestellt werden.

Ein Dokumentenbaum stellt einen geordneten und in der Anzahl der Kindknoten – also direkten Nachfolgern eines Knotens – unbegrenzten Baum dar, der jedem Element und jedem Elementtext eines Dokuments einen Knoten zuordnet. Das Wurzelement eines Dokuments bildet dabei die Wurzel des Dokumentenbaums. Eine Kante innerhalb eines Dokumentenbaums von Knoten p nach Knoten p' existiert, falls p der Elternknoten – also der direkte Vorgänger – von p' ist, d.h. falls das zu p' gehörende Element direkt in dem zu p gehörende Ele-

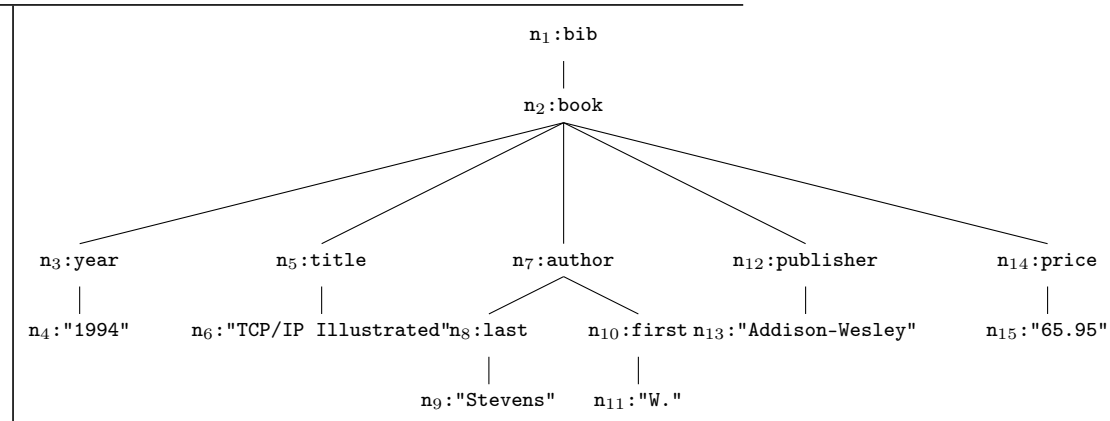
¹Eine alphanumerische Zeichenkette ist dabei im weiten Sinne zu verstehen, d.h. nicht nur aus Buchstaben und Zahlen bestehend, sondern auch inklusive von Sonderzeichen (z.B. „“ etc.), so lange sich diese Zeichen im vom XML-Standard verwendeten Vorrat des ISO/IEC 10646 Zeichensatzes befinden und nicht Teil des (geschützten) XML eigenen Markups (z.B. „<“ etc.) sind.

ment enthalten ist oder p' für den Elementtext von p steht. Nachfolgendes Beispiel 2.2 stellt für das bib-Wurzelement und das erste book-Element aus Quelltext B.1, also für die Zeichenkette² aus Beispiel 2.1, den ihm zugehörigen Dokumentenbaum dar. Innerhalb dieses Dokumentenbaums stellen die Knoten der Menge $N_{elem} = \{n_1, n_2, n_3, n_5, n_7, n_8, n_{10}, n_{12}, n_{14}\}$ *Elementknoten* und Knoten der Menge $N_{text} = \{n_4, n_6, n_9, n_{11}, n_{13}, n_{15}\}$ *Textknoten* dar.

Beispiel 2.1 (XML-Teildokument von Quelltext B.1)

```
<bib>
  <book>
    <year>1994</year>
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
</bib>
```

Beispiel 2.2 (Dokumentenbaum für Beispiel 2.1)



Eine *Erweiterung* eines Dokumentenbaums ist gegeben, falls neben Element- und Textknoten auch die Attribute innerhalb des öffnenden Tags eines Elements als Kindknoten, d.h. als direkte Nachfolger von diesem Elementknoten als *Attributknoten*, in den Dokumentenbaum eingefügt werden. Aus diesem Grund werden, da somit keine Einschränkung der Dokumente vorliegt, in der weiteren Arbeit nur solche Dokumente betrachtet, deren öffnender Tag der Elemente keine Attribute besit-

²Innerhalb der gesamten Arbeit sind in den Beispielen von Dokumenten sämtliche Leerzeichen, Tabs etc., die, wie in Abschnitt „3.7.1.4 Boundary Whitespace“ in [36] beschrieben wird, als PCDATA interpretiert werden und somit zu Textknoten des entsprechenden Dokumentenbaums führen, zu ignorieren und dienen lediglich der besseren Übersicht und Lesbarkeit.

zen. Die u.U. weiteren in einem Dokumentenbaum existierenden *Knotentypen*, wie bspw. *Kommentarknoten*, die aus den in einem Dokument stehenden Kommentaren stammen, kommen ebenfalls bei den in dieser Arbeit betrachteten Dokumenten nicht vor und werden ignoriert.

Durch die zugrunde gelegte Ordnung auf einem Dokumentenbaum kann mit Hilfe der *Tiefensuche* (*left-to-right depth-first search*) oder des *Tiefendurchlaufs* (*left-to-right depth-first traversal*) in der Auswahlreihenfolge der Nachfolger eines Knotens von links nach rechts das einem Dokumentenbaum zugehörige Dokument samt (korrekter) *Dokumentordnung*, d.h. (korrekter) Reihenfolge der öffnenden Tags, zurückgewonnen werden.

Der Zugang zu Inhalt und Struktur eines Dokuments wird meist mit Hilfe eines *XML-Prozessors* ermöglicht. Dieser besitzt dazu meist einen *XML-Parser*, der den Zugriff auf Inhalt und Struktur eines Dokuments für den XML-Prozessor bereitstellt. Zwei der bekanntesten XML-Parser sind *Document Object Model (DOM)* und *Simple API for XML (SAX)*.

Ein *DOM-Parser* stellt ein Dokument in Form eines Baums, welcher – bis auf die Wurzel – dem Dokumentenbaum des Dokuments entspricht, dar. Die Wurzel des durch den DOM-Parser gewonnenen (DOM-)Baums bildet die *Dokumentwurzel*, welche ein virtueller (Element-)Knoten³ ist und sich *über* dem Wurzelement des Dokuments befindet und somit dessen direkter Vorgänger ist. Neben dem Wurzelement eines Dokuments kann die Dokumentwurzel des (DOM-)Baums weitere Kinder, die aus Kommentaren, Verarbeitungsanweisungen etc. eines Dokumentes stammen, besitzen. Trotz der Erweiterung eines Dokumentenbaums um die Dokumentwurzel zum Erhalt eines (DOM-)Baums werden der (DOM-)Baum und der Dokumentenbaum in dieser Arbeit als gleichwertig angesehen und nicht unterschieden, wobei meist unter einem Dokumentenbaum der (DOM-)Baum verstanden wird. Daher wird auch bei der Darstellung von Dokumentenbäumen stets auf die Dokumentwurzel – da sie Bestandteil eines jeden (DOM-)Baums ist – verzichtet.

Der *SAX-Parser* ist gegenüber dem DOM-Parser ereignisorientiert und bearbeitet ein Dokument als sequenziellen Datenstrom, sprich in Form seiner Zeichenkette, und führt bei Auftreten bestimmter Ereignisse, z.B. eines öffnenden oder schließen-

³Virtueller (Element-)Knoten bedeutet, dass die Dokumentwurzel eines (DOM-)Baums zwar real existiert, allerdings *nur* auf das Wurzelement bzw. auf die eigenen Kinder verweist.

den Tags, eine vorgegebene Funktion oder Aktion aus. Dadurch wird ermöglicht, bei Auftreten bestimmter Ereignisse oder Daten entsprechende Aktionen zu tätigen bzw. dem aktuellen Stand der Verarbeitung eines Dokuments entsprechend zu reagieren.

2.3 XML Path Language (XPath)

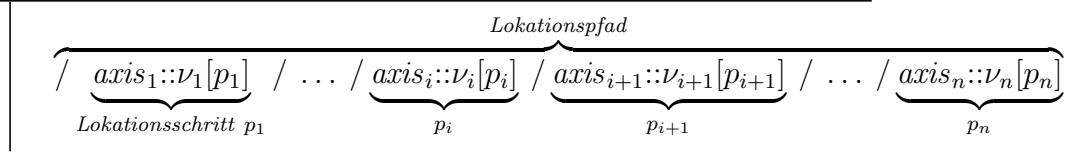
Eine Möglichkeit zum Adressieren bzw. Lokalisieren von Teilen eines Dokuments, das dazu aus der Sicht seines Dokumentenbaums (inklusive der Dokumentwurzel) betrachtet wird, bietet *XML Path Language (XPath)* durch die Verwendung von Pfaden. Ein *Pfadausdruck* in XPath, *Lokationspfad* (kurz: Pfad/-e) genannt, besteht aus einer durch „/“ getrennten Folge von *Lokationsschritten* (kurz: Schritt/-e) der Form $axis::\nu[p]$, wobei *axis* für eine *Achse*, ν für einen *Knotentest* und die optionale Angabe p für ein oder mehrere *Prädikat/-e* stehen. Ein solcher Pfad ist dabei *absolut*, falls ihm „/“ vorangestellt ist, ansonsten *relativ*. Absolut bedeutet, dass der Pfad bei der Dokumentwurzel des Dokumentenbaums beginnt und diese – da sie auf das Wurzelement verweist – den gesamten Dokumentenbaum repräsentiert. Im Gegensatz dazu bezieht sich ein relativer Pfad auf einen *Kontextknoten*, also einen aktuellen bzw. im Moment betrachteten Knoten eines Dokumentenbaums.

Eine Folge von Schritten eines Pfades wird dabei induktiv von links nach rechts ausgewertet, wobei ein Schritt für den ihm folgenden Schritt die zu betrachtende Menge von Kontextknoten⁴ für diesen festlegt. Besteht also ein Pfad P , wie mit Beispiel 2.3 verdeutlicht wird, aus $n \geq 1$ Schritten p und gilt $1 \leq i \leq n$, so lokalisiert Schritt p_i eine Menge von Knoten⁵, die die zu betrachtenden Kontextknoten für den darauf folgenden Schritt p_{i+1} sind.

⁴Dabei ist natürlich zu beachten, dass die durch einen Schritt lokalisierte Knotenmenge eines Dokumentenbaums die Vereinigung aller lokalisierten Knoten eines Schrittes und somit stets ohne Duplikate ist. Dieser Sachverhalt, dass ein Schritt identische Knoten eines Dokumentenbaums aus Sicht zweier oder mehrerer unterschiedlicher Kontextknoten des ihm vorangehenden Schrittes lokalisiert, kann bspw. durch die Verwendung der Achse „descendant“ und gleichem oder ineinander enthaltenem Knotentest zweier oder mehrerer nacheinander folgenden Schritten auftreten.

⁵Aufgrund der in dieser Arbeit stark baumorientierten Sicht auf ein Dokument wird auf das im XPath-Standard 1.0 in [33] verwendete Wort „Knotenmenge“ (node-set) und nicht das im XPath-Standard 2.0 in [34] dafür eingeführte Wort einer „Sequenz von Items“ (sequence of items) zurückgegriffen.

Beispiel 2.3 ((Lokations-)Pfad und (Lokations-)Schritte)



Die durch den letzten Schritt p_n dieser Folge lokalisierte Menge von Knoten legt das Resultat für den gesamten Pfad fest.

Mit Hilfe der *Achsen*, wie sie in der nachfolgenden Tabelle 2.1 aufgelistet sind, wird innerhalb der einzelnen Schritte angegeben, in welcher Weise der Dokumentenbaum (weiter) durchlaufen bzw. in welche „Richtung“ dieser traversiert werden soll.

Achse	Definierte Menge von Knoten
attribute (Attribute)	Alle Attribute des Kontextknotens.
child (Kinder)	Alle direkten Nachfolger des Kontextknotens.
descendant (Nachkommen)	Alle direkten und indirekten Nachfolger des Kontextknotens.
descendant-or-self (Nachkommen plus Kontextknoten)	Wie „descendant“, jedoch einschließlich des Kontextknotens.
parent (Eltern)	Der direkte Vorgänger des Kontextknotens.
ancestor (Vorfahren)	Alle Knoten, außer dem Kontextknoten, auf dem Pfad von diesem zum repräsentierenden Knoten des Wurzelements.
ancestor-or-self (Vorfahren plus Kontextknoten)	Wie „ancestor“, jedoch einschließlich des Kontextknotens.
following (nachfolgende Knoten)	Alle Knoten, die in der Dokumentordnung nach dem Kontextknoten stehen, jedoch außer den durch „descendant“ definierten Knoten.
following-sibling (nachfolgende Geschwister)	Alle Knoten, die denselben direkten Vorgänger haben wie der Kontextknoten und in der Dokumentordnung nach diesem stehen.
preceding (vorherige Knoten)	Alle Knoten, die in der Dokumentordnung vor dem Kontextknoten stehen, jedoch außer den durch „ancestor“ definierten Knoten.
preceding-sibling (vorherige Geschwister)	Alle Knoten, die denselben direkten Vorgänger haben wie der Kontextknoten und in der Dokumentordnung vor diesem stehen.
self (Kontextknoten)	Aktueller Knoten.

Tabelle 2.1: XPath-Achsen und lokalisierte Knotenmengen nach [11]

Tabelle B.1 verdeutlicht für Beispiel 2.1 nochmals die erhaltenen Knoten der einzelnen Achsen aus Sicht des *author*-Elementknotens, d.h. mit dem *author*-Elementknoten als Kontextknoten des ersten Schrittes.

Die Achsen können in *Vorwärts-* oder *Rückwärtsachsen*, je nachdem, welche Knoten bzw. Knotenmengen von ihnen lokalisiert werden, unterteilt werden. Eine Achse ist nach [34] eine Vorwärtsachse, falls sie nur Knoten lokalisiert, die vom Kontextknoten aus ihm in der Dokumentordnung folgen. Entsprechend ist sie eine Rückwärtsachse, falls sie nur Knoten lokalisiert, die vom Kontextknoten aus ihm in der Dokumentordnung vorausgehen. So zählen *attribute*, *child*, *descendant*, *descendant-or-self*, *following* und *following-sibling* zu den Vorwärtsachsen und *parent*, *ancestor*, *ancestor-or-self*, *preceding* und *preceding-sibling* zu den Rückwärtsachsen. Die Achse *self*, die die Kontextknoten eines vorangehenden Schrittes lokalisiert, ist sowohl Vorwärts- als auch Rückwärtsachse. Durch Ausnutzung der *Achsensymmetrien*, z.B. der Achse „parent“ und der Achse „child“, ist es nach [9] möglich, einen Pfad mit Rückwärtsachsen in einen äquivalenten Pfad mit ausschließlich Vorwärtsachsen zu überführen. Ebenfalls ist es aus Sicht eines beliebigen Kontextknotens möglich, mit Hilfe der Achsen *self*, *ancestor*, *descendant*, *preceding* und *following* durch mengenmäßige Vereinigung der erhaltenen Knotenmengen alle Knoten eines Dokumentenbaums und somit das gesamte Dokument, das durch ihn repräsentiert wird, zu erfassen.

Der *Knotentest* eines Schrittes kann genutzt werden, um den Knotentyp oder den Namen der zu selektierenden Knoten aus der durch die Achse lokalisierten Knotenmenge festzulegen. So können aus dieser Menge durch die Angabe eines *Elementnamens* alle Elementknoten mit diesem Namen, durch „*“ alle Element- oder Attributknoten, durch „*text()*“ alle Textknoten oder durch „*node()*“ alle Knoten, d.h. ohne Einschränkung des Knotentyps, extrahiert werden.

Die Angabe von *Prädikaten* innerhalb eines Schrittes dienen der zusätzlichen Filterung, um das Ergebnis bzw. die erhaltene Knotenmenge weiter einzuschränken. Dabei werden die Knoten, die sich aus der Achse und dem Knotentest ergeben, selektiert, für die der XPath-Ausdruck, der als Prädikat angegeben ist, einem Wahrheitswert zugeordnet werden kann und dieser „true“ ergibt. Dazu können neben Pfaden in Prädikaten Vergleichsoperatoren (<, <=, =, !=, >=, >), logische Operatoren (*and*, *or*, ! etc.), arithmetische Operatoren (+, −, *, *div*, *mod* etc.)

sowie eine Reihe von Funktionen (*sum*, *avg*, *min*, *max*, *count* etc.) zur Selektion genutzt werden. Ein Pfad, der eine nicht leere Menge von Knoten lokalisiert, besitzt dazu den Wert „true“, andernfalls den Wert „false“.⁶ Pfade, die als Prädikat in Kombination mit einem Vergleichsoperator verwendet werden, werden, falls der durch den Pfad lokalisierte Knoten kein Textknoten ist, durch die Konkatenation der einzelnen Werte aller nach diesem lokalisierten Knoten befindlichen Textknoten des Dokumentenbaums zu einem einzigen Wert zusammengefasst. Ebenfalls ist es durch die Angabe eines Prädikates möglich, einen Knoten anhand dessen *Kontextposition* in der durch die Achse und den Knotentest lokalisierten Knotenmenge auszuwählen. Die Größe der lokalisierten Knotenmenge, die *Kontextgröße*, ist dabei durch ihre Mächtigkeit gegeben.

Im Weiteren entsprechen verkürzende Schreibweisen innerhalb von Pfaden den gewöhnlichen Abkürzungen von XPath. So steht bspw. `//title` für `/descendant::title`, `/bib//title` für `/child::bib/descendant::title`⁷ und `/bib/book[1]` für `/bib/book[position() = 1]`. Weiter wird die Achse „descendant-or-self“ durch die Kurzform „dos“, d.h. `/bib/descendant-or-self::node()/title` durch `/bib/dos::node()/title`, abgekürzt.

2.4 XML Query Language (XQuery)

XPath bildet die Grundlage für *XML Query Language (XQuery)* von [36], die u.a. das Ziel verfolgt, Elemente bzw. Teile aus einem Dokument zu extrahieren und diese u.U. in ein neues (erzeugtes) Dokument einzubetten oder schlicht auszugeben.

Eine *XQuery-Anfrage* (kurz: Anfrage/-n) besteht im Wesentlichen aus den drei Teilen *Versionsdeklaration*, *Prolog* und keinem, einem oder mehreren *XQuery-Ausdrücken* (kurz: Ausdruck/Ausdrücke). Die ersten beiden Teile sind dabei optional und werden hier, wie die Möglichkeit, eine Anfrage mit Kommentaren zu versehen, außer Acht gelassen. Ein häufig in einer Anfrage⁸ verwendeter Ausdruck, wie

⁶Dieser Sachverhalt ist nur für den XPath-Standard 1.0 in [33] zutreffend.

⁷Nach [34] wird während der Verarbeitung eines Pfades jedes nicht initiale `//` effektiv durch `/descendant-or-self::node()/` ersetzt. Somit ergibt der Pfad `/bib//title` mit vollständig ausgeschriebenem Schritten `/child::bib/descendant-or-self::node()/child::title`. Allerdings können in manchen Fällen Pfade wie z.B. `/child::bib/descendant::title` zu `/bib//title` verkürzt werden.

⁸Wie für Dokumente, gilt auch für alle Beispiele von Anfragen innerhalb dieser Arbeit, dass Leerzeichen, Tabs etc., wie in Abschnitt „3.7.1.4 Boundary Whitespace“ in [36] beschrieben, zu ignorieren sind und nur aufgrund der besseren Übersicht und Lesbarkeit eingefügt wurden.

in Beispiel 2.4 dargestellt, ist der *FLWOR-Ausdruck*. Dabei steht FLWOR für die Anfangsbuchstaben der Klauseln *for*, *let*, *where*, *order by* und *return*, die im Allgemeinen eine Anfrage enthält.

Beispiel 2.4 (*FLWOR-Ausdruck für Quelltext B.1*)

Die nachfolgende Anfrage gibt alle Buchtitel des Autors mit Nachnamen „Stevens“ aus. Die Sortierung und somit die Ausgabe der Buchtitel erfolgt dabei in aufsteigender Reihenfolge.

```
<r> {  
  for $title in /bib/book/title  
    let $author := $title/following-sibling::author  
    let $ttext := $title/text()  
    where ($author/last = "Stevens")  
    order by $ttext ascending return  
      <book_title> {$ttext} </book_title>  
} </r>
```

Resultat:

```
<r>  
  <book_title>Advanced Programming In The Unix Environment</book_title>  
  <book_title>TCP/IP Illustrated</book_title>  
</r>
```

Nach [12] wird durch eine *for*-Klausel eine oder auch mehrere Variable/-n an Ausdrücke gebunden und erzeugen dadurch einen Strom von Tupeln. Dieser, für das Erzeugen des Stroms verantwortliche Ausdruck, ist meist ein Pfad, dessen resultierende bzw. lokalisierte Knotenmenge⁹ Knoten für Knoten sukzessive an die Variable gebunden wird und dadurch eine *for*-Klausel iterieren lässt. Der Zugriff auf den an eine Variable gebundenen Knoten in anderen Ausdrücken erfolgt durch die Verwendung der Variablen, in dem diese einem Pfad vorangestellt wird. Im Unterschied zur *for*-Klausel erlaubt es die *let*-Klausel, eine oder mehrere Variable/-n jeweils an ein gesamtes Resultat eines Ausdrucks zu binden. Eine Einschränkung bzw. Filterung der Tupel der *for*-Klausel wird durch die *where*-Klausel erreicht, wohingegen eine Sortierung der Tupel durch die *order by*-Klausel erfolgen kann. Ist keine Sortierung angegeben, so erfolgt die Auswahl bzw. Ausgabe der Tupel in Dokumentordnung. Die Rück- bzw. Ausgabe und somit das Resultat eines FLWOR-Ausdrucks wird mit der *return*-Klausel definiert. Dabei werden die Werte bzw. Tupel des Stroms

⁹Wie bereits für XPath wird in dieser Arbeit aufgrund der stark baumorientierten Sicht auf ein Dokument nicht das im XQuery-Standard 1.0 in [36] verwendete Wort einer „Sequenz von Items“ (sequence of items), sondern dafür das Wort „Knotenmenge“ (node-set) verwendet.

zurück- bzw. ausgegeben, die durch den nach der return-Klausel stehenden Ausdruck bestimmt werden.

Neben der Verwendung von FLWOR-Ausdrücken können innerhalb von Ausdrücken einer Anfrage eine Reihe von Standardfunktionen wie Aggregatfunktionen (*sum*, *avg*, *min*, *max*, *count*) und diverse weitere, bedingte Ausdrücke, d.h. if-then-else Konstruktionen oder Funktionen wie „exists“, die auf das Vorhandensein eines Knotens im Dokumentenbaum prüft, verwendet werden. Dazu stehen besonders für bedingte Ausdrücke oder innerhalb von where-Klauseln, wie bereits für XPath, Vergleichsoperatoren, logische Operatoren oder auch arithmetische Operatoren zur Verfügung. Für eine ausführliche Definition der XQuery Anfragesprache und die formale Definition der Semantik wird an dieser Stelle auf [36] bzw. [37] verwiesen.

Kapitel 3

Aktive Speicherbereinigung: Garbage Collected XQuery (GCX)

Dieser Abschnitt befasst sich mit der in [20] bzw. [21] vorgestellten *aktiven Speicherbereinigung*. Dort wird der Ausgangspunkt für deren Erweiterungen und Optimierungen beschrieben. Die aktive Speicherbereinigung selbst ist eine Kombination aus *statischer* und *dynamischer Analyse* bei der Auswertung von Anfragen auf Dokumente, die als Datenstrom erhalten werden. Damit wird versucht, den Zielen (1) nur relevante (Dokument-)Daten, die zur Auswertung einer Anfrage benötigt werden, zu speichern¹, (2) (Dokument-)Daten während der Auswertung einer Anfrage nicht länger im Speicher zu belassen als notwendig und (3) während der Auswertung einer Anfrage keine Mehrfachkopien derselben (Dokument-)Daten im Speicher zu erhalten, näher zu kommen. Aus den Zielen (1) bis (3) lässt sich konstatieren, dass das primäre Ziel in der Reduzierung bzw. Minimierung des Speicherbedarfs liegt.

Eine praktische Umsetzung der aktiven Speicherbereinigung erfolgte in [20] bzw. [21] in der „*Garbage Collection XQuery*“ (GCX) Engine in Form eines Prototypen für ein Sprachfragment von XQuery. Diese unterstreicht deutlich eine Verringerung des Speicherbedarfs durch die Kombination von statischer und dynamischer Analyse, was sich zusätzlich in einer kürzeren Auswertungszeit einer Anfrage äußert.²

Im nun Folgenden wird zunächst das *Sprachfragment XQ* beschrieben, das den Aufbau und die Ausdrücke einer Anfrage der aktiven Speicherbereinigung bzw. der GCX Engine festlegt. Von diesem ausgehend werden die Aufgaben der statischen

¹Speichern ist innerhalb dieser Arbeit eher als Zwischenspeichern zu interpretieren. (Dokument-) Daten, die gespeichert werden, liegen nur temporär, solange eine Anfrage ausgewertet wird, vor.

²s. dazu „Experimental Results“ in [20] oder [21].

und dynamischen Analyse, sowohl theoretisch durch Definitionen als auch zur Verdeutlichung anhand von Beispielen erklärt, und die wesentlichen Charakteristika der GCX Engine aufgeführt. Dabei sind Teile, wie bspw. Definitionen, im Ganzen als auch teilweise aus [20] bzw. [21] übernommen und ins Deutsche übersetzt worden.

3.1 Sprachfragment XQ

Die *Syntax des Sprachfragments XQ* ist in Abbildung 3.1 dargestellt und legt den Aufbau einer Anfrage und die in einer Anfrage verwendbaren Ausdrücke fest. Dabei spezifiziert es (1) die Einbettung von beliebigem (gemischtem) Elementinhalt oder Ausdrücken in Elementkonstruktoren zur Erzeugung neuer Elemente bzw. die Verwendung dieser innerhalb von Ausdrücken, (2) die Bildung von Sequenzen von beliebigem (gemischtem) Elementinhalt oder von Ausdrücken, (3) die unterstützten Klauseln eines FLWOR-Ausdrucks, (4) die Verwendung bedingter Ausdrücke und die darin unterstützten Funktionen, logischen Operatoren und Vergleichsoperatoren sowie (5) die unterstützten Achsen und Knotentests in Pfaden.

$XQuery ::= \epsilon \mid XMLExpr$
 $XMLExpr ::= \langle QName \rangle NestedXMLExpr \langle / QName \rangle$
 $\quad \mid \langle QName \rangle \langle / QName \rangle \mid \langle QName / \rangle$
 $NestedXMLExpr ::= \{ QExpr \} \mid String \mid XMLExpr \mid NestedXMLExpr NestedXMLExpr$
 $QExpr ::= ReturnQExpr \mid QExpr, QExpr$
 $ReturnQExpr ::= QExprSingle \mid (QExpr) \mid ()$
 $QExprSingle ::= \text{“String”} \mid FWRExpr \mid IfExpr \mid VarExpr \mid NestedXMLExpr$
 $FWRExpr ::= ForClause [\mathbf{where} Condition]? \mathbf{return} ReturnQExpr$
 $ForClause ::= \mathbf{for} \$ VarName \mathbf{in} VarAxisExpr [, \$ VarName \mathbf{in} VarAxisExpr]^*$
 $IfExpr ::= \mathbf{if} (Condition) \mathbf{then} ReturnQExpr \mathbf{else} ReturnQExpr$
 $Condition ::= \mathbf{fn:true}() \mid \mathbf{fn:false}() \mid \mathbf{fn:exists}(VarAxisExpr)$
 $\quad \mid (Condition) \mid Operand RelOp Operand \mid \mathbf{fn:not}(Condition)$
 $\quad \mid Condition \mathbf{and} Condition \mid Condition \mathbf{or} Condition$
 $Operand ::= VarExpr \mid \text{“String”}$
 $RelOp ::= < \mid \leq \mid \geq \mid > \mid = \mid \neq$
 $VarExpr ::= \$ VarName \mid VarAxisExpr$
 $VarAxisExpr ::= \$ VarName PathStepExpr \mid PathStepExpr$
 $PathStepExpr ::= / \mid Axis NodeTest$
 $Axis ::= / \mid // \mid \mathbf{child::} \mid // \mid \mathbf{descendant::}$
 $NodeTest ::= \mathbf{node}() \mid \mathbf{text}() \mid * \mid QName$

$QName ::=$ Elementname

$String ::=$ Zeichenkette

$VarName ::=$ XQuery Variable (z.B. $\$x$, $\$y$, ...)

Abbildung 3.1: XQuery Sprachfragment XQ aus [4]

Das Sprachfragment XQ ist nach [3] *kompositionsfrei* (*composition-free*), was sich im Verlust der *let-Klausel* des *FLWOR-Ausdrucks* äußert. Allerdings können nach [3] in vielen Fällen *let-Klauseln* aus einer Anfrage entfernt bzw. umgeschrieben werden und somit Anfragen, die zuvor *let-Klauseln* enthielten, an das Sprachfragment XQ angepasst werden. Neben der *let-Klausel* findet auch die *order by-Klausel* keine Unterstützung. Die *where-Klausel* und deren darin enthaltene(n) Bedingung(en), des somit verbleibenden *FWR-Ausdrucks* des Sprachfragments XQ, wird/werden bei Verwendung in einer Anfrage, wie in [1, 6] beschrieben, durch die *Konstruktion* von bedingten Ausdrücken ersetzt. Zu beachten ist jedoch, um die *Wohlgeformtheit* des Resultats bzw. des ausgegebenen Dokuments zu gewährleisten, dass die beiden Bedingungen rechts und links eines logischen Operators in *Bedingungen*, die in der *where-Klausel* eines *FWR-Ausdrucks* oder in bedingten Ausdrücken einer Anfrage des Sprachfragments XQ auftreten, *syntaktisch gleich* sind. Dies bedeutet, dass keine Elemente bzw. deren öffnende und schließende Tags, mit dem logischen Operator zwischen diesen konstruiert werden dürfen. Dementsprechend können Anfragen, die beliebig verschachtelte *FWR-Ausdrücke*, bedingte Ausdrücke und Joins enthalten, durch das Sprachfragment XQ gebildet werden. Die Bildung von Joins erfolgt dabei intern durch (naive) nested loop joins.

Eine Einschränkung des Sprachfragments XQ besteht allerdings darin, dass *alle Pfade* in einer Anfrage bzw. innerhalb deren Ausdrücke nur jeweils *einen Schritt* besitzen dürfen. Nach [1, 6] besteht zumindest für Pfade von *for-Klauseln*, die mehrfache Schritte besitzen, die Möglichkeit, diese eventuell, durch ineinander verschachtelte *for-Klauseln* mit Pfaden, die nur einen Schritt besitzen, auszudrücken. Die einzelnen Pfade können dazu für diesen einen Schritt die *vorwärtsgerichteten Achsen* „*child*“ und „*descendant*“ sowie für den *Knotentest* die Angabe eines *Elementnamens*, „***“, „*text()*“ und „*node()*“ verwenden. Die Angabe bzw. Verwendung von *Prädikaten* in diesem Schritt eines Pfades findet hingegen innerhalb des Sprachfragments XQ keine Unterstützung. Somit kann, wie in [3] argumentiert wird, ein Großteil der in der Praxis auftretenden Anfragen abgedeckt werden. Dies gilt allerdings nur eingeschränkt für den Vergleich von (Element-)Knoten anhand der Konkatenation der Inhalte aller Textknoten ihrer Teilbäume zu einer (alphanume-

rischen) Zeichenkette³ und ohne Verwendung von Aggregatfunktionen.

Die *Semantik des Sprachfragments XQ* ist dabei mit der von XQuery⁴ deckungsgleich. Der *Auswertungsablauf* einer Anfrage in XQ erfolgt aufgrund der Auswertung einer Anfrage auf ein als Datenstrom erhaltenes Dokument *strikt sequenziell*. Das bedeutet, dass die Auswertung einer Anfrage (Ausdruck für Ausdruck) top-down, dem in der weiteren Arbeit noch besondere Aufmerksamkeit⁵ zukommt, stattfindet. So ist die Auswertung eines Ausdrucks α mit k freien Variablen einer Anfrage in XQ über eine Funktion $[[\alpha]]_k$ definiert, die ein k -Tupel von Knoten als Eingabe (z.B. eine momentane Bindung der k Variablen an Knoten) besitzt. In dieser Funktion stehen k für die Anzahl gebundener Variablen, \vec{e} für einen Zeiger der momentanen Bindungen der Variablen an Knoten, das Symbol \uplus für die Listenkonkatenation und l_i für das i -te Element der Liste l . Demnach wird ein FR-Ausdruck sequenziell zu einer Liste, die Elementknoten und/oder Textknoten enthält, und wie folgt evaluiert wird.

$$[[\text{for } \$x_{k+1} \text{ in } \$y/\text{axis}::\nu \text{ return } \beta]]_k(\vec{e}) := \biguplus_{1 \leq i \leq |l|} [[\beta]]_{k+1}(\vec{e}, l_i) \text{ mit } l = [[\$y/\text{axis}::\nu]]_k(\vec{e}).$$

Dabei wird eine Variable, z.B. $\$x_{k+1}$, sukzessive an jeden Knoten in der Liste, die der Knotenmenge entspricht, die durch den Pfad „ $\$y/\text{axis}::\nu$ “ lokalisiert wird, gebunden und der Körper der return-Klausel bzw. der nach ihr stehende Ausdruck wird sofort nach einer neuen Bindung dieser Variablen neu ausgewertet. Für eine vollständige Beschreibung und Übersicht der *Evaluierungsstrategie* des Sprachfragments XQ wird an dieser Stelle auf [3] verwiesen.

In der weiteren Arbeit wird auf das Präfix „fn:“ von Funktionen des Sprachfragments XQ, z.B. „fn:exists“, in den noch folgenden Anfragen verzichtet.

³Im Falle, dass eine Variable an einen Elementknoten gebunden ist bzw. ein Pfad einen Elementknoten lokalisiert ergibt sich die (alphanumerische) Zeichenkette zum Vergleich aus der Konkatenation aller nach diesem Elementknoten befindlichen Inhalte der Textknoten eines Dokumentenbaums. Dabei wird nicht verglichen, ob die beiden (Element-)Knoten des Dokumentenbaums identisch sind.

⁴s. dafür [37]

⁵Die wichtige Bedeutung des Auswertungsablauf liegt an den in eine Anfrage eingefügten SignOff-Anweisungen, deren Ausführung auf gerade diesem Ablauf der Auswertung beruhen.

3.2 Statische Analyse

Die *statische Analyse* erfolgt zur *Übersetzungszeit* (*compile-time*), also noch *vor Auswertung* einer Anfrage bzw. *vor der Verarbeitung* des als Datenstrom erhaltenen Dokuments. Sie besitzt im Wesentlichen zwei Aufgaben, die die Grundsteine für die *dynamische Analyse* sind.

Von ihr wird der für eine Anfrage zugehörige *Projektionsbaum* erstellt, der die Teile eines Dokuments festlegt, die zur Auswertung einer Anfrage benötigt werden. Dabei kommt die in [1, 29, 31] beschriebene *Dokumentprojektion* zum Einsatz, die eine etablierte Technik zur Speicherminimierung ist. Mit Hilfe dieser und unter Verwendung des einer Anfrage zugehörigen Projektionsbaums werden die zur Auswertung einer Anfrage benötigten und somit relevanten Teile eines Dokuments bestimmt und auch nur diese gespeichert. Nicht benötigte und somit irrelevante Teile eines Dokuments werden dabei verworfen und nicht gespeichert.

Die zweite wesentliche Aufgabe der statischen Analyse ist das Einfügen von *SignOff-Anweisungen* in eine Anfrage, die signalisieren, dass bereits gespeicherte Teile eines Dokuments nicht mehr benötigt werden, also ihre Relevanz in Bezug auf die (weitere) Auswertung einer Anfrage verloren haben, und (wieder) aus dem Speicher entfernt werden können.

3.2.1 Funktion und Herleitung eines Projektionsbaums

Das Speichern eines von einem Datenstrom stammenden Dokuments in vollem Umfang vor Auswertung einer Anfrage kann u.U. aufgrund beschränkter Ressourcen nicht vollständig erfolgen. Aus diesem Grund ist es notwendig so früh als möglich nicht benötigte Teile eines Dokuments, die für das Resultat bzw. die Auswertung einer Anfrage keine Bedeutung besitzen zu verwerfen und nur die Teile zu extrahieren und zu speichern, die für eine Anfrage relevant sind. Eine Technik mit der dies erreicht werden kann ist die in [1, 29, 31] beschriebene *Dokumentprojektion*, die die zur Auswertung einer Anfrage benötigten Teile eines Dokuments bestimmt, wohingegen nicht benötigte Teile verworfen werden. Die Auswertung einer Anfrage wird daraufhin auf dem (projizierten) Dokument ausgeführt.

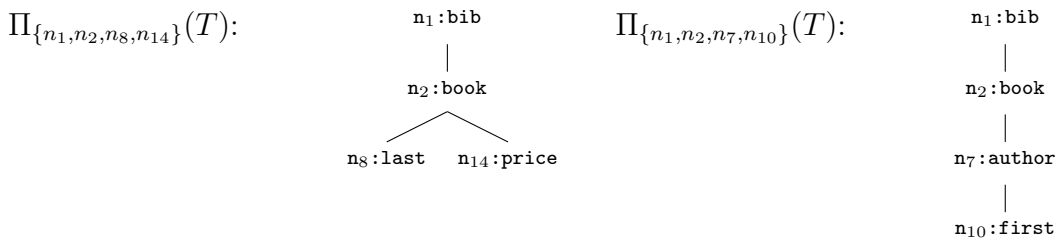
Die Bestimmung der zur Auswertung einer Anfrage relevanten Teile eines Dokuments erfolgt mit Hilfe von *Projektionspfaden*. Diese werden aus den in einer Anfrage enthaltenen Pfaden hergeleitet. Dabei werden nach [1, 31], unter Darstellung des Dokuments als Dokumentenbaum, Knoten, die sich in der Knotenmenge befinden, die durch einen Projektionspfad lokalisiert wird, samt all ihrer Vorfahren gespeichert. Die Speicherung der Vorfahren relevanter Knoten ist allerdings nicht immer erforderlich. Mit Hilfe der nachfolgenden Definition 3.1 wird es in der aktiven Speicherbereinigung ebenfalls möglich, auch die Vorfahren relevanter Knoten, falls diese nicht benötigt werden, zu entfernen. Dadurch kann die Größe des (projizierten) Dokuments nochmals reduziert werden. Das hat besonders dann Auswirkungen, wenn Pfade in einer Anfrage an den Nachkommen eines Knotens interessiert sind, d.h. die Achse „descendant“ in einem Schritt enthalten ist.

Definition 3.1 (Dokumentprojektion $\Pi_S(T)$)

Sei T ein Dokumentenbaum und dom die Menge der Knoten in T . Sei weiter $S \subseteq dom$ eine Knotenmenge mit $root \in S$. Die (*Dokument-*)*Projektion von T in Bezug auf S* , bezeichnet mit $\Pi_S(T)$, ist der Dokumentenbaum bestehend aus der Knotenmenge S und den Vorfahren-Nachkommen- und nachfolgende Knoten-Beziehungen wie in T .

Beispiel 3.1 (Dokumentprojektion $\Pi_S(T)$ an Beispiel 2.2)

Für den Dokumentenbaum T aus Beispiel 2.2 ergibt sich:



Die im nachfolgenden Beispiel 3.2 enthaltene Anfrage bildet die Grundlage weiterer Beispiele in diesem Kapitel.

Beispiel 3.2 (Anfrage für Quelltext B.1)

Die nachfolgende Anfrage gibt alle Bücher aus, die keinen Preis besitzen. Darauf folgend werden alle Buchtitel ausgegeben.

```
<r> {
  for $bib in /bib return
    ((for $x in $bib/* return
      if (not(exists($x/price))) then $x else ()),
     for $b in $bib/book return $b/title)
} </r>
```

Wie nachfolgendes Beispiel 3.3 verdeutlicht, besitzen Anfragen in XQ, die absolute Pfade enthalten und somit Knoten aus Sicht der Dokumentwurzel lokalisieren, implizit die Variable $\$root$ diesen (absoluten) Pfaden vorangestellt. Diese Variable besitzt in den Definitionen zwar eine Bedeutung, sie ist aber für die weiteren Anfragen in den Beispielen weggelassen worden.

Beispiel 3.3 (Anfrage aus Beispiel 3.2 mit Variable $\$root$)

Die in Beispiel 3.2 enthaltene Anfrage ergibt die nachfolgend dargestellte Anfrage mit Variable $\$root$ den (absoluten) Pfaden vorangestellt.

```
<r> {
  for $bib in $root/bib return
    ((for $x in $bib/* return
      if (not(exists($x/price))) then $x else ()),
     for $b in $bib/book return $b/title)
} </r>
```

Für das Herleiten der Projektionspfade, die die relevanten Teile eines Dokuments für eine Anfrage festlegen, werden zunächst die nachfolgenden Definition 3.2, Definition 3.3 und Definition 3.4 benötigt.

Definition 3.2 (Variablenmenge $Vars_Q$)

Die Menge der in einer Anfrage Q auftretenden Variablen inklusive der Variablen $\$root$ wird mit $Vars_Q$ bezeichnet.

Beispiel 3.4 (Variablenmenge $Vars_Q$ für Beispiel 3.2)

Für die Anfrage Q aus Beispiel 3.2 ergibt sich $Vars_Q = \{\$root, \$bib, \$x, \$b\}$.

Definition 3.3 (Unterausdruck $\alpha \preceq \beta$)

Für zwei Ausdrücke α und β einer Anfrage Q wird mit $\alpha \preceq \beta$ bzw. $\alpha \prec \beta$ bezeichnet, dass α ein *Unterausdruck* bzw. *echter Unterausdruck* von β ist.

Beispiel 3.5 (Unterausdrücke $\alpha \preceq \beta$ für Beispiel 3.2)

Sei mit $for_{\$z}$ die for-Klausel bezeichnet, die Variable $\$z$ einführt, dann ergibt sich für die Anfrage Q aus Beispiel 3.2 $for_{\$x} \prec for_{\$bib} \prec Q$ und $for_{\$b} \prec for_{\$bib} \prec Q$.

Definition 3.4 (Rollenfunktion $r_Q(\beta) = r$)

Sei die *Rollenfunktion* $r_Q : XQ \rightarrow Rollen$ eine injektive Funktion, die jedem Ausdruck einer Anfrage in XQ eine eindeutige Rolle zuweist.

Beispiel 3.6 (Rollenfunktion $r_Q(\beta) = r$ für Beispiel 3.2)

Sei $\beta_1 = „exists(\$x/price)“$, $\beta_2 = „\$x“$ und $\beta_3 = „\$b/title“$, dann wäre für die Anfrage Q aus Beispiel 3.2 eine mögliche Rollenfunktion $r_Q(\beta_1) = r_2$, $r_Q(\beta_2) = r_3$ und $r_Q(\beta_3) = r_5$.

Das Herleiten der Projektionspfade aus einer Anfrage erfolgt mit Hilfe von Definition 3.5. Nach dieser Definition ist es in Bedingungen in denen mit der Funktion „exists“ auf das Vorhandensein von Knoten geprüft wird, ausreichend, nur den ersten Knoten, der durch den in der Funktion stehenden Pfad lokalisiert wird, zu speichern, da alle weiteren durch diesen Pfad lokalisierten Knoten für das Ergebnis bzw. den Wahrheitswert der Funktion keine Relevanz besitzen. Knoten, die von Pfaden in Ausgaben oder in bedingten Ausdrücken lokalisiert werden, werden zusammen mit all ihren Nachkommen gespeichert, während Knoten, die zur Iteration einer for-Klausel benötigt werden ohne ihre jeweiligen Teilbäume, d.h. nur die lokalisierten Knoten selbst, gespeichert werden.

Definition 3.5 (Abhängigkeiten $dep_Q(\$x)$)

Sei Q eine Anfrage in XQ und $\$x \in Vars_Q$. Die Menge der *Abhängigkeiten der Variablen* $\$x$, bezeichnet mit $dep_Q(\$x)$, ist eine Menge von Tupeln der Form $\langle \$x/\pi, r \rangle$ mit einer Variablen $\$x$, einem (relativen) Pfad π und einer Rolle r . Mit anderen Worten beinhalten die Abhängigkeiten Pfade relativ zur Bindung einer Variablen $\$x$ und sind wie folgt definiert. Sei $\beta \preceq Q$ mit Rollenfunktion $r_Q(\beta) = r$, dann

- $\langle axis::\nu[1], r \rangle \in dep_Q(\$x)$, falls $\beta = \text{„exists}(\$x/axis::\nu\text{“}$,
- $\langle axis::\nu/dos::node(), r \rangle \in dep_Q(\$x)$, falls β entweder ein Ausdruck in einer Ausgabe der Form $\text{„}\$x/axis::\nu\text{“}$, ein bedingter Ausdruck der Form $\text{„}\$x/axis::\nu \text{ RelOp } \chi\text{“}$ oder $\text{„}\chi \text{ RelOp } \$x/axis::\nu\text{“}$, und
- $\langle dos::node(), r \rangle \in dep_Q(\$x)$, falls β ein Ausdruck in einer Ausgabe der Form $\text{„}\$x\text{“}$.

Beispiel 3.7 (Abhängigkeiten $dep_Q(\$x)$ für Beispiel 3.2)

Seien innerhalb der einzelnen Abhängigkeiten die Rollen r_i aus Beispiel 3.6, dann ergibt sich für die Anfrage Q aus Beispiel 3.2

$$dep_Q(\$x) = \{ \langle child::price[1], r_2 \rangle, \langle dos::node(), r_3 \rangle \}$$

$$dep_Q(\$b) = \{ \langle child::title/dos::node(), r_5 \rangle \}.$$

Alle weiteren Variablen, $\$root$ und $\$bib$, der Menge $Vars_Q$ besitzen keine Abhängigkeiten.

Die dadurch erhaltenen *Projektionspfade*, die zur Projektion eines Dokuments genutzt werden, können in einem *Projektionsbaum* zusammengefasst werden. In diesem Baum stellt jeder Pfad von der Wurzel zu einem Knoten einen Projektionspfad dar, d.h. jeder Knoten beschreibt einen Pfad, der die relevanten Teile eines Dokuments für eine Anfrage bestimmt.

Die Konstruktion eines Projektionsbaums erfolgt unter Zuhilfenahme des einer Anfrage zugehörigen *Variablenbaums*. Dieser ist, wie in Beispiel 3.9 dargestellt, ein ungeordneter und in der Anzahl der Kindknoten eines Knotens unbegrenzter Baum,

mit dem die Eltern-Kind-Beziehungen zwischen den Variablen einer Anfrage ausgedrückt werden. Er selbst ist über die Knotenmenge $Vars_Q$ und Kantenbeziehungen der Elternvariable $parVar_Q$, die in Definition 3.6 festgelegt werden, definiert.

Definition 3.6 (Elternvariable $parVar_Q(\$x) = \y)

Für zwei Variablen $\$x, \$y \in Vars_Q$ ist die Elternvariable $\$y$ von $\$x$, geschrieben $parVar_Q(\$x) = \y , wie folgt definiert:

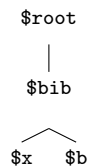
$$parVar_Q(\$x) \stackrel{def}{=} \begin{cases} \$root, & \text{falls } \beta = \text{„for } \$x \text{ in } /axis::\nu \text{ return } \alpha\text{“} \\ \$y, & \text{falls } \beta = \text{„for } \$x \text{ in } \$y /axis::\nu \text{ return } \alpha\text{“} \\ undefined, & \text{falls } \$x = \$root. \end{cases}$$

Beispiel 3.8 (Elternvariablen $parVar_Q(\$x) = \y für Beispiel 3.2)

Für die Anfrage Q aus Beispiel 3.2 ergibt sich
 $parVar_Q(\$root) = undefined$, $parVar_Q(\$bib) = \$root$,
 $parVar_Q(\$x) = \bib und $parVar_Q(\$b) = \bib .

Beispiel 3.9 (Variablenbaum für Beispiel 3.2)

Für die Anfrage aus Beispiel 3.2 ergibt sich der nachfolgend dargestellte Variablenbaum.



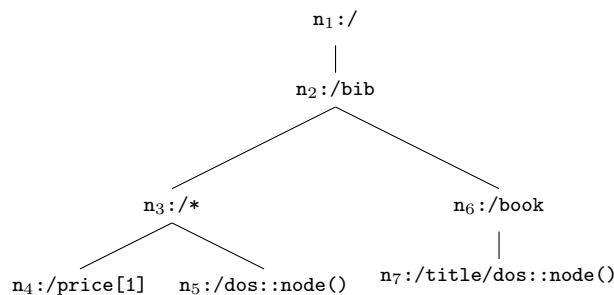
Ein Projektionsbaum der aus einem Variablenbaum konstruiert wird, ist formal ein ungeordneter und in der Anzahl der Kindknoten eines Knotens unbegrenzter Baum. Seine Wurzel ist mit „/“ beschriftet, mit dem kenntlich gemacht wird, dass alle (Projektions-)Pfade, die durch einen Knoten in diesem beschrieben werden, absolut sind. Alle seine inneren Knoten sind mit Pfaden (mit einem Schritt) der Form „/axis:: ν “ beschriftet.

Zur Gewinnung des Projektionsbaums aus dem Variablenbaum erhält zunächst der Wurzelknoten, $\$root$, des Variablenbaums die Beschriftung „/“, mit dem kenntlich gemacht wird, dass alle durch einen Knoten des Projektionsbaums beschrie-

bene Pfade absolut sind. Danach wird der Variablenbaum um Knoten erweitert, die mit Pfaden beschriftet sind. So wird für jede Variable $\$x$ und für jede Abhängigkeit $\langle \$x/\pi, r \rangle \in dep_Q(\$x)$ ein Knoten n mit Beschriftung „/ π “ und eine Kante von $\$x$ nach n in den Variablenbaum eingefügt. Diesen Knoten wird mit Hilfe der Rollenfunktion $r_Q(\beta)$ die Rolle $r_\pi(n) = r$ vergeben. Weiter wird jeder mit einer Variablen, $\$x$, benannter Knoten n des Variablenbaums mit dem Pfad seiner zugehörigen for-Klausel $\beta = \text{„for } \$x \text{ in } \$y/axis::\nu \text{ return } \alpha\text{“}$, über dessen Knotenmenge diese iteriert, ersetzt, d.h. die Variable $\$x$ wird durch „/ $axis::\nu$ “ ersetzt und diesem Knoten ebenfalls mit Hilfe der Rollenfunktion $r_Q(\beta)$ die Rolle $r_\pi(n) = r_Q(\beta)$ vergeben. Als Resultat erhält man, wie nachfolgendes Beispiel 3.10 zeigt, den einer Anfrage zugehörigen Projektionsbaum.

Beispiel 3.10 (Projektionsbaum für Beispiel 3.2)

Für die Anfrage aus Beispiel 3.2 ergibt sich der nachfolgend dargestellte Projektionsbaum.



Die ebenfalls den Knoten eines Projektionsbaums vergebenen (eindeutigen) Rollen werden den Teilen eines Dokuments bzw. der den zu speichernden Knoten dieses Dokuments zugehörigen Dokumentenbaums vor ihrer Speicherung von der dynamischen Analyse zugewiesen. Diese Rollen sind eng mit der Dokumentprojektion verbunden und bilden die Grundlage der von der dynamischen Analyse ausgeführten Speicherbereinigung. Rollen sind eine endliche Menge von Elementen. Eine Rollenmenge ist eine Multimenge von Rollen und als Funktion $m : Rollen \rightarrow \mathbb{IN}$ definiert, die die (einzelnen) Rollen auf ihre Vielfachheit innerhalb der Rollenmenge abbildet. Dabei bezeichnet die Vielfachheit null einer Rolle, dass diese nicht in der Rollenmenge enthalten ist und die leere Rollenmenge, bezeichnet mit \emptyset , dass alle Rollen in der Rollenmenge die Vielfachheit null besitzen.

Diese Rollenmengen⁶ werden von der dynamischen Analyse den zu speichernden Knoten eines Dokumentenbaums zugewiesen. Hierbei ist zu beachten, dass nur zu speichernde Elementknoten eines Dokumentenbaums Rollen erhalten können. Diese den zu speichernden Knoten eines Dokumentenbaums zugewiesenen Rollen dienen als Indikator für die Relevanz dieser bei der Auswertung der (weiteren) Anfrage. Das Prinzip, den zu speichernden Knoten Rollen zuzuweisen, ist verwandt mit dem Zählen von Referenzen (reference counting), die ein Objekt besitzt. Diese Technik ist eine der in [22] beschriebenen weit verbreitete Technik in Programmiersprachen zur automatisierten Speicherverwaltung. Das Ziel, das damit verfolgt wird, ist „tote“ Objekte zu identifizieren und diese zu entfernen, von denen sicher ist, dass auf sie in der Zukunft nicht mehr zugegriffen wird. Beim Zählen von Referenzen (reference counting) geschieht dies anhand eines Zählers, der für jedes Objekt, die auf ihn zeigenden Referenzen zählt. Der Zähler wird erhöht, falls eine Referenz hinzukommt und verringert nach Entfernen einer Referenz. Falls der Zähler null wird, kann das Objekt, da nicht mehr darauf zugegriffen werden kann, gelöscht werden und der von diesem Objekt belegte Speicherbedarf (wieder) freigegeben werden.

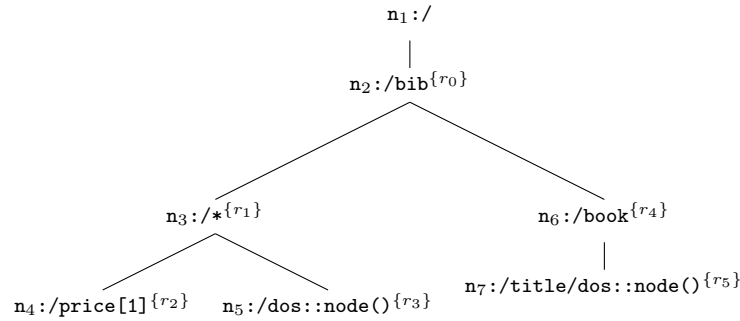
Ähnlich diesem Prinzip wird jedem zu speichernden Knoten eines Dokumentenbaums mit Hilfe der *Rollenzuweisungsfunktion* $\rho : \text{dom} \rightarrow m$ vor seiner Speicherung eine gewisse Anzahl einer Rolle zugewiesen. Aus ihr wird die Vielfachheit m der (einzelnen) Rollen, die ein zu speichernder Knoten erhält, gewonnen. Dabei erhält jeder zu speichernde Knoten Rollen zugewiesen, im Normalfall mindestens eine Rolle. Weiter sind zwei Funktionen, $\text{add}_\rho(r, n)$ und $\text{rem}_\rho(r, n)$, hergeleitet, die für einen Knoten n und Rolle r Rollen einem Knoten zuweisen bzw. entfernen. Formal bedeutet dies, falls $\rho(n) = m$ und $m(r) = i$ gilt, so ist nach Ausführung von $\text{add}_\rho(r, n)$, $\rho(n)(r) = i + 1$ sowie nach Ausführung von $\text{rem}_\rho(r, n)$, $\rho(n)(r) = i - 1$, falls $i > 0$ und undefiniert, falls $i = 0$.

Daraus ergibt sich der im nachfolgenden Beispiel 3.11 dargestellte Projektionsbaum mit Rollen, wobei dessen Wurzel keine Rolle erhält.

⁶Im Weiteren wird die den zu speichernden Knoten eines Dokumentenbaums zugewiesene Rollenmenge nur mit Rollen bezeichnet.

Beispiel 3.11 (Projektionsbaum mit Rollen für Beispiel 3.2)

Für die Anfrage aus Beispiel 3.2 ergibt sich der nachfolgend dargestellte Projektionsbaum mit Rollen.



3.2.2 Aufgabe und Einfügen von SignOff-Anweisungen

Die zweite Aufgabe der statischen Analyse ist das „Umschreiben“ einer Anfrage. Dabei werden *SignOff-Anweisungen* in diese eingefügt, die der *dynamischen Analyse* signalisieren, dass Rollen an den gespeicherten Knoten entfernt werden können, da diese ihre Relevanz für die (weitere) Anfrage verloren haben. Dabei kann die in Satz 3.1 gemachte Aussage, dass die Auswertung einer umgeschriebenen Anfrage mit SignOff-Anweisungen auf einem projiziertem Dokumentenbaum das gleiche Resultat liefert wie die Auswertung einer Anfrage ohne SignOff-Anweisungen auf einem nicht projiziertem Dokumentenbaum, garantiert werden.

Satz 3.1 (Korrektheit bei der Auswertung einer Anfrage)

Sei Q eine Anfrage in XQ und T der zu verarbeitende Dokumentenbaum. Sei weiter Q' die umgeschriebene Anfrage von Q mit SignOff-Anweisungen und T' der projizierte Dokumentenbaum mit zugewiesenen Rollen an seinen Knoten. Dann gilt: $[[Q]]_1(T) = [[Q']]_1(T')$.

Bevor SignOff-Anweisungen allerdings eingefügt werden können, bedürfen *bedingte Ausdrücke* einer besonderen Behandlung. Diese können bis zu diesem Zeitpunkt, da noch keine Auswertung der Anfrage und noch keine Verarbeitung des als Datenstrom erhaltenen Dokuments erfolgt ist, nicht entschieden werden. Grundsätzlich stehen SignOff-Anweisungen niemals in der Bedingung eines bedingten Ausdrucks, sondern stets am Ende des Gültigkeitsbereichs einer Variablen, d.h. als zuletzt

auszuführende Anweisung(en) einer for-Klauseln. Daher werden, falls eine Anfrage *bedingte Ausdrücke* enthält und diese im *then-* oder *else-*Fall eine for-Klausel besitzen, diese durch die in Abbildung 3.2 genannten *Inferenzregeln* „umgeformt“. Dabei wird versucht, bedingte Ausdrücke in for-Klauseln „hineinzudrücken“, indem zuerst die Inferenzregel *Zerlegung* (*ZL*) an jeden bedingten Ausdruck der Anfrage angewandt wird. Die daraus erhaltene Anfrage, die nur „leere“ else-Fälle enthält, wird darauf durch beliebiges Anwenden der Regeln *Sequenz* (*SQ*), *Elementkonstruktor* (*EK*) und *Hineindrücken* (*HD*) bis zu einem Fixpunkt weiter umgeformt.

$$\begin{aligned} \text{ZL: } & \frac{\text{if } X \text{ then } \alpha \text{ else } \beta}{(\text{if } X \text{ then } \alpha \text{ else } (), \text{ if } (\text{not } X) \text{ then } \beta \text{ else } ())} \\ \text{SQ: } & \frac{\text{if } X \text{ then } (\alpha_1, \dots, \alpha_n) \text{ else } ()}{(\text{if } X \text{ then } \alpha_1 \text{ else } (), \dots, \text{if } X \text{ then } \alpha_n \text{ else } ())} \\ \text{EK: } & \frac{\text{if } X \text{ then } \langle a \rangle \alpha \langle /a \rangle \text{ else } ()}{(\text{if } X \text{ then } \langle a \rangle \text{ else } (), \text{if } X \text{ then } \alpha \text{ else } (), \text{if } X \text{ then } \langle /a \rangle \text{ else } ())} \\ \text{HD: } & \frac{\text{if } X \text{ then for } \$x \text{ in } \$y/\text{axis}::\nu \text{ return } \alpha \text{ else } ()}{\text{for } \$x \text{ in } \$y/\text{axis}::\nu \text{ return if } X \text{ then } \alpha \text{ else } ()} \end{aligned}$$

Abbildung 3.2: Inferenzregeln zum „Hineindrücken“ von bedingten Ausdrücken in FR-Ausdrücke aus [21]

Beispiel 3.12 („Hineindrücken“ von bedingten Ausdrücken in FR-Ausdrücke)

Die nachfolgende Anfrage gibt, falls Bücher existieren, eine Liste aller Titel, sonst den Tag `<no-books/>` aus.

```
<r> {
  for $bib in /bib return
    if (exists($bib/book)) then
      <book-list> {
        for $book in $bib/book return
          <book> {$book/title} </book>
      } </book-list>
    else <no-books/>
} </r>
```

Die Anwendung der *Zerlegungsregel* (ZL) liefert nachfolgendes Ergebnis.

```
<r> {
  for $bib in /bib return
    (
      if (exists($bib/book)) then
        <book-list> {
          for $book in $bib/book return
            <book> {$book/title} </book>
        } </book-list>
      else (),
      if (not(exists($bib/book))) then <no-books/> else ()
    )
} </r>
```

Die Anwendung der *Elementkonstruktor-* (EK) und *Hineindrückenregel* (HD) in dieser Reihenfolge an den ersten bedingten Ausdruck liefert folgendes Endergebnis.

```
<r> {
  for $bib in /bib return
    (
      (
        if (exists($bib/book)) then <book-list> else (),
        for $book in $bib/book return
          if (exists($bib/book)) then <book> {$book/title} </book> else (),
        if (exists($bib/book)) then </book-list> else ()
      ),
      if (not(exists($bib/book))) then <no-books/> else ()
    )
} </r>
```

Diese Art der „Umformung“ ist eher theoretisch, da sie in der Praxis aufgrund des Erzeugens vieler identischer Ausdrücke ineffizient ist. In der praktischen Umsetzung innerhalb der GCX Engine ist deshalb eine andere Strategie zum Einsatz gekommen. Dazu wird während der Auswertung einer Anfrage zunächst die Bedingung geprüft und, abhängig von dieser, der then- oder else-Fall ausgeführt. Nachdem

dies erfolgt ist, wird der nicht eingetretene Fall ohne Ausgabe, d.h. nur die SignOff-Anweisungen werden ausgeführt, ausgewertet. Das dadurch erhaltene Resultat einer Anfrage wird durch diese Strategie nicht beeinflusst oder verfälscht.

Eine *SignOff-Anweisung* bezieht sich auf einen oder mehrere gespeicherte Knoten und stets auf genau eine Rolle. Dabei sollten SignOff-Anweisungen in der Anfrage (1) so früh wie möglich stehen, um den Speicherbedarf möglichst gering zu halten, und dürfen (2) während der Auswertung einer Anfrage nicht zu früh ausgeführt werden, um das Resultat dieser durch Entfernen von zur Auswertung einer Anfrage noch benötigten und somit relevanten Knoten nicht zu verfälschen.

Die *Syntax* einer SignOff-Anweisung wird mit der nachfolgenden Definition 3.7 festgelegt. Die *Semantik* einer SignOff-Anweisung ist allerdings, wie Definition 3.13 festlegt, Bestandteil der dynamischen Analyse, da sie für das Löschen der Rollen zuständig ist und dies während der Laufzeit, d.h. während der Auswertung einer Anfrage, erfolgt.

Definition 3.7 (*Syntax einer SignOff-Anweisung*)

Eine *SignOff-Anweisung* ist ein Ausdruck der Form $\text{signOff}(\$x/\pi, r)$, mit einer Variablen $\$x$, einem (relativen) Pfad π , der auch leer sein kann, und einer Rolle r .

Mit Hilfe der nachfolgenden Definition 3.8 wird sichergestellt, dass alle Knoten, die Rollen erhalten haben, diese auch während der Auswertung der Anfrage wieder verlieren. Dies ist spätestens nach (vollständiger) Auswertung der Anfrage der Fall.

Definition 3.8 (*Sicherheit bei der Auswertung mit SignOff-Anweisungen*)

Die *Auswertung einer umgeschriebenen Anfrage mit SignOff-Anweisungen* auf ein Dokument ist *sicher*, falls die folgenden Bedingungen gelten:

1. Alle Rollen, die zur Laufzeit entfernt werden, sind definiert.
2. Nachdem die Anfrage ausgewertet wurde, sind alle Rollen entfernt worden.

Bei der Ausführung einer SignOff-Anweisung verlieren am Ende des Geltungsbereiches einer jeden Variablen $\$x$ alle Knoten, die von $\$x$ abhängen und für die $\$x$ die erste direkte Vorgängervariable ist, wie in Definition 3.10 und Definition 3.11 beschrieben ist, ihre zugewiesenen Rollen.

Definition 3.9 (Vorgängervariable $\$x <_Q \y)

Für zwei Variablen $\$x, \$y \in Vars_Q$ ist $\$y$ die *Vorgängervariable* von $\$x$, geschrieben $\$x <_Q \y , falls entweder (1) $\$y = parVar_Q(\$x)$ oder (2) es eine Variable $\$z$ gibt, so dass $\$x <_Q \z und $\$z <_Q \y existiert, gilt. Die Kurzschreibweise $\$x \leq_Q \y steht entweder für $\$x =_Q \y oder $\$x <_Q \y .

Beispiel 3.13 (Vorgängervariablen $\$x <_Q \y für Beispiel 3.2)

Für die Anfrage Q aus Beispiel 3.2 ergibt sich
 $\$bib <_Q \$root$, $\$x <_Q \bib und $\$b <_Q \bib .

Definition 3.10 (Direkte Variable)

Sei Q eine Anfrage in XQ und $\$z \in Vars_Q$. Die Variable $\$z$ ist *direkt*, falls entweder $\$z = \$root$ oder ein FR-Ausdruck der Form $\beta = \text{„for } \$z \text{ in } \$y/axis::\nu \text{ return } \alpha\text{“}$, so dass (1) $\$y$ direkt und (2) kein FR-Ausdruck der Form $\gamma = \text{„for } \$u \text{ in } \$v/axis'::\nu' \text{ return } \alpha\text{“}$ in dem $\$u$ keine Vorgängervariable von $\$z$ und $\beta \prec \gamma \preceq Q$ existiert, gilt.

Beispiel 3.14 (Direkte Variablen)

Für die Anfrage aus Beispiel 3.2 sind die Variablen $\$root$, $\$bib$, $\$x$ und $\$b$ direkt.

Betrachtet man allerdings die nachfolgende Anfrage, die alle Bücher ausgibt,

```
<r> {
  for $bib in //bib return
    <bib> {
      for $book in //book return $book
    } </bib>
} </r>
```

dann sind die Variablen $\$root$ und $\$bib$ direkt und $\$book$ allerdings nicht.

Definition 3.11 (Erste direkte Vorgängervariable $fsa_Q(\$x)$)

Sei Q eine Anfrage und $\$x \in Vars_Q$. Die *erste direkte Vorgängervariable* (*first straight ancestor* (fsa)) von $\$x$, bezeichnet mit $fsa_Q(\$x)$, ist wie folgt definiert

$$fsa_Q(\$x) \stackrel{def}{=} \begin{cases} \$x, & \text{falls } \$x \text{ direkt} \\ fsa_Q(parVar_Q(\$x)), & \text{sonst.} \end{cases}$$

Beispiel 3.15 (Erste direkte Vorgängervariablen $fsa_Q(\$x)$)

Für die Anfrage Q_1 aus Beispiel 3.2 sind die Variablen $\$root$, $\$bib$, $\$x$ und $\$b$, daher ergibt sich

$$fsa_{Q_1}(\$root) = \$root, fsa_{Q_1}(\$bib) = \$bib, fsa_{Q_1}(\$x) = \$x \text{ und } fsa_{Q_1}(\$b) = \$b.$$

Betrachtet man allerdings die Anfrage Q_2 aus Beispiel 3.14, bei der die Variablen $\$root$ und $\$bib$ direkt sind und $\$book$ allerdings nicht, dann ergibt sich

$$fsa_{Q_2}(\$root) = \$root, fsa_{Q_2}(\$bib) = \$bib \text{ und } fsa_{Q_2}(\$book) = \$root.$$

Das Einfügen von SignOff-Anweisungen in eine Anfrage erfolgt mit Hilfe der in Abbildung 3.3 genannten Inferenzregeln. Die erste Regel fügt für die Variable $\$root$ die entsprechenden SignOff-Anweisungen ein. Die zweite Regel fügt diese für die verbleibenden Variablen immer am Ende der sie einführenden for-Klausel ein.

$$\beta = Q: \frac{\beta : \langle a \rangle \alpha \langle /a \rangle}{\langle a \rangle (\alpha, signOffInsert_Q(\$root)) \langle /a \rangle}$$

$$\beta \prec Q: \frac{\beta : \{\text{for } \$x \text{ in } \$y/\sigma \text{ return } \alpha\}}{\{\text{for } \$x \text{ in } \$y/\sigma \text{ return } (\alpha, signOffInsert_Q(\$x))\}}$$

Abbildung 3.3: Inferenzregeln zum Einfügen von SignOff-Anweisungen in eine Anfrage aus [21]

Zum Einfügen der SignOff-Anweisungen in eine Anfrage wird dazu der in Quelltext 3.1 beschriebenen Algorithmus $su_Q(\$x)$ verwendet. Dieser benötigt nachfolgende Definition 3.12.

Definition 3.12 (Variablenpfad $varPath_Q(\$y, \$x)$)

Für zwei Variablen $\$x \leq_Q \y ist der *Variablenpfad* zwischen $\$y$ und $\$x$, geschrieben $varPath_Q(\$y, \$x)$, wie folgt rekursiv definiert. Sei $\$z$ eine Variable, so dass $\$x \leq_Q \$z <_Q \$y$ mit einem FR-Ausdruck in Q der Form „for $\$z$ in $\$y/axis::\nu$ return α “, dann ist

$$varPath_Q(\$y, \$x) \stackrel{def}{=} \begin{cases} \epsilon, & \text{falls } \$x = \$y \\ axis::\nu / varPath_Q(\$z, \$x), & \text{sonst.} \end{cases}$$

Beispiel 3.16 (Variablenpfade $varPath_Q(\$y, \$x)$)

Für die Anfrage Q_1 aus Beispiel 3.2 ergibt sich

$varPath_{Q_1}(\$root, \$bib) = child::book$, $varPath_{Q_1}(\$bib, \$x) = child::*$ und $varPath_{Q_1}(\$bib, \$b) = child::book$.

Betrachtet man allerdings die nachfolgende Anfrage Q_2 , die alle Bücher, gefolgt von allen Titeln, ausgibt,

```
<r> {
  for $bib in /bib return
  (
    for $book in $bib/book return $book,
    for $title in $bib//title return $title
  )
} </r>
```

dann ergibt sich

$varPath_{Q_2}(\$root, \$bib) = child::book$, $varPath_{Q_2}(\$bib, \$book) = child::book$ und $varPath_{Q_2}(\$bib, \$title) = descendant::title$.


```

// Algorithmus signOffInsertQ(Variable  $\$x$ )
begin
  if ( $\$x \neq \$root$ ) then
    begin
      Sei  $\$x$  durch den FR-Ausdruck  $\beta = \text{„for } \$x \text{ in } \$y/axis::\nu \text{ return } \alpha\text{“}$  definiert;
      emit „signOff( $\$x, r_Q(\beta)$ )“; // Ausgabe der entsprechenden SignOff-Anweisung
    end
  for each Variable  $\$z \in Vars_Q$ , so dass  $fsa_Q(\$z) = \$x$ 
  begin
    Sei  $\sigma = varPath_Q(\$x, \$z)$ ;
    for each  $\langle \pi, r \rangle \in dep_Q(\$z)$ 
    begin
      emit „signOff( $\$x/\sigma/\pi, r$ )“; // Ausgabe der entsprechenden SignOff-Anweisung(en)
    end
  end
end
end

```

Quelltext 3.1: Algorithmus *signOffInsert_Q*($\$x$) zum Einfügen von SignOff-Anweisungen aus [21]

Die nach dem Umschrieben resultierende Anfrage mit SignOff-Anweisungen ist im nachfolgenden Beispiel 3.17 dargestellt.

Beispiel 3.17 (Anfrage aus Beispiel 3.2 mit SignOff-Anweisungen)

Für die Anfrage aus Beispiel 3.2 ergibt sich die nachfolgende (umgeschriebene) Anfrage mit SignOff-Anweisungen.

```

<r> {
  for $bib in /bib return
    ((for $x in $bib/* return
      (if (not(exists($x/price))) then $x else (),
        signOff($x, r1),
        signOff($x/price[1], r2),
        signOff($x/dos::node(), r3))),
    (for $b in $bib/book return
      ($b/title,
        signOff($b, r4),
        signOff($b/title/dos::node(), r5))),
    signOff($bib, r0))
} </r>

```

3.3 Dynamische Analyse

Die *dynamische Analyse* erfolgt im Gegensatz zur statischen Analyse zur *Laufzeit* (*runtime*), also *während* der *Auswertung* einer Anfrage und *während* der *Verarbeitung* des als Datenstrom erhaltenen Dokuments. Sie besitzt, wie auch die statische

Analyse, im Wesentlichen zwei Aufgaben. So ist sie zum einen für das *Zuweisen* von *Rollen* den zu speichernden Knoten und dem *Entfernen* dieser an den bereits gespeicherten Knoten zuständig. Des Weiteren wird von ihr die *Ausführung* der *Speicherbereinigung* vollzogen. Bei der Speicherbereinigung werden bereits gespeicherte Knoten, die zur Auswertung dieser nicht benötigt werden, während der Auswertung einer Anfrage aus dem Speicher entfernt, um den von ihnen belegten Speicherbedarf wieder freizugeben.

3.3.1 Zuweisen und Entfernen von Rollen

Vor der Speicherung von Knoten eines Dokumentenbaums werden diesen Rollen zugewiesen, die als Indikator für deren Relevanz bei der Auswertung der Anfrage bzw. eines Ausdrucks dienen. Bei der Zuweisung von Rollen, die bei der Verarbeitung bzw. der Projektion des Dokuments erfolgt, kann es möglich sein, dass ein Knoten mehrere Rollen erhält, falls er in den Knotenmengen enthalten ist, die durch mehrere beschriebenen Pfade eines Projektionsbaums lokalisiert werden. Darüber hinaus kann ein Knoten auch eine Rolle mehrfach erhalten, falls z.B. der Schritt eines Pfades in einer Anfrage die Achse „descendant“ enthält. Erst nachdem ein Knoten eine Rolle zugewiesen bekommen hat, wird dieser gespeichert.

Beim Entfernen der Rollen spielen die zuvor von der statischen Analyse in die Anfrage eingefügten SignOff-Anweisungen eine wesentliche Rolle, da sie Hand in Hand mit den vergebenen Rollen der bereits gespeicherten Knoten arbeiten. Diese signalisieren, dass bestimmte Knoten ihre Relevanz für die Auswertung der Anfrage verloren haben. Dem liegt die Annahme zugrunde, dass die Anzahl anfangs zugewiesener Rollen für jeden gespeicherten Knoten und die Anzahl SignOff-Anweisungen, die während der Auswertung ausgeführt werden, übereinstimmen.

Beim Entfernen der Rollen an den gespeicherten Knoten ist die *Semantik* einer SignOff-Anweisung, wie Definition 3.13 festlegt, von Bedeutung. Diese bestimmt das Vorgehen bei der Ausführung einer SignOff-Anweisung.

Definition 3.13 (*Semantik einer SignOff-Anweisung*)

Sei T ein Dokumentenbaum und ρ die Rollenzuweisungsfunktion. Sei weiter \vec{e} die momentanen Bindungen von k Variablen an Knoten, $\$x$ eine Variable in \vec{e} und r eine Rolle, dann ist die *Semantik* einer *SignOff-Anweisung*

$$[[\text{signOff}(\$x/\pi, r)]]_k(\vec{e})$$

wie folgt

1. Definiere zunächst eine Knotenmenge S . Sei x der Knoten an den die Variable $\$x$ im Moment gebunden ist, d.h. $x = [[\$x]]_k(\vec{e})$. Falls $\pi = \epsilon$, so ist $S = \{x\}$. Andernfalls ist S die von Kontextknoten x nach Auswertung des Pfades π lokalisiert Menge von Knoten.
2. Entferne Rolle r (durch das Ausführen von $\text{rem}_\rho(n, r)$) von allen Knoten $n \in S$.

Aus praktischer Sicht erfolgt das eigentliche Entfernen von Rollen an den gespeicherten Knoten mit Hilfe des in Quelltext 3.2 enthaltenen nachfolgenden Algorithmus, der im Wesentlichen der Semantik einer SignOff-Anweisung aus Definition 3.13 folgt. Dieser vollzieht auch parallel dazu die Speicherbereinigung, die allerdings Aufgabe der dynamischen Analyse ist und daher erst im nächsten Unterabschnitt 3.3.2 beschrieben wird.

```
// Algorithmus signOff($x/π, Rolle r)
begin
  Sei x der Knoten, an den die Variable $x im Moment gebunden ist;
  Sei weiter die Knotenmenge S wie folgt definiert:
    if ( $\pi = \epsilon$ ) then  $S := \{x\}$ 
    else  $S := \{\text{von Kontextknoten } x \text{ nach Auswertung des Pfades } \pi \text{ lokalisierte Menge von Knoten}\}$ 
  for each Knoten  $n \in S$ 
    begin
      execute  $rem_{\rho}(r, n)$  // Entferne Rolle r von Knoten in S
      while ( $n \neq root$  and  $n$  ist irrelevant) // Lokale Suche
        begin
          Sei  $p$  der Elternknoten von  $n$ ;
          if ( $n$  ist beendet) then loesche  $n$ ; // schliessender Tag bereits gelesen
          else markiere  $n$  als geloescht // schliessender Tag (noch) nicht gelesen
            und loesche  $n$  unmittelbar nachdem sein schliessender Tag gelesen wurde;
           $n := p$ ; // bottom-up
        end
      end
    end
  end
```

Quelltext 3.2: Algorithmus *signOff*(\$x/π, r) zum Ausführen einer SignOff-Anweisung aus [21]

3.3.2 Ausführen der Speicherbereinigung

Neben dem Entfernen von Rollen durch den Algorithmus in Quelltext 3.2 wird auch die Speicherbereinigung durch diesen vollzogen. Diese dient dem Entfernen von bereits gespeicherten Knoten, die nach Definition 3.14 irrelevant sind, also keine Relevanz für die Auswertung der weiteren Anfrage mehr besitzen. Nach dieser Definition können Knoten, die alle ihre Rollen verloren haben und die keine Nachkommen, die eine Rolle inne haben, besitzen, bereits während der Laufzeit – ohne die Auswertung der (weiteren) Anfrage zu beeinflussen oder das Resultat dieser zu verfälschen – (wieder) aus dem Speicher entfernt werden und den von ihnen belegten Speicherbedarf freigeben.

Definition 3.14 (*Irrelevante Knoten*)

Ein (gespeicherter) Knoten x eines Dokuments bzw. seines zugehörigen Dokumentenbaums ist *irrelevant*, falls für ihn und alle Nachkommen y von x die Menge der ihnen zugewiesenen Rollen leer ist, d.h. weder x noch y eine Rolle besitzen.

Die zuvor in die Anfrage eingefügten SignOff-Anweisungen dienen dem *aktiven* Anstoßen der Speicherbereinigung, die dadurch so früh als möglich irrelevante Knoten aus dem Speicher wieder entfernen. Die Speicherbereinigung wird dabei jedes Mal nachdem ein Knoten eine Rolle verloren hat ausgeführt.

Der Dokumentenbaum wird, falls ein Knoten gelöscht wird weiter *bottom-up* und solange irrelevante Knoten durch das bottom-up gefunden werden, durchlaufen. Dies kann sich bis zur Dokumentwurzel des Dokumentenbaums fortsetzen. Besondere Beachtung muss dabei auf Knoten gelegt werden, deren schließender Tag noch nicht gelesen wurde.⁷ Diese werden, um die Auswertung bzw. das Resultat einer Anfrage nicht zu beeinflussen oder zu verfälschen, nicht sofort gelöscht, sondern so lange im Speicher behalten, bis ihr schließender Tag gelesen wurde, und erst daraufhin entfernt.

3.4 Praktische Umsetzung: Garbage Collected XQuery (GCX)

Eine praktische Umsetzung der *aktiven Speicherbereinigung*, d.h. die Kombination von *statischer* und *dynamischer Analyse*, ist in der „*Garbage Collected XQuery*“ (GCX) Engine in Form eines Prototypen für das Sprachfragment XQ erfolgt. Als Programmiersprache ist C++ verwendet worden, da sie, im Gegensatz zu anderen Programmiersprachen (z.B. Java), keine automatische Speicherbereinigung besitzt, sondern eine direkte Kontrolle über die Speicherzuweisung und -freigabe ermöglicht. Diese Eigenschaft, direkten Einfluss auf die Speicherverwaltung zu besitzen, ist Grundvoraussetzung für die Umsetzung der aktiven Speicherbereinigung, im Besonderen für die der dynamischen Analyse.

⁷Dazu erhalten alle gespeicherten Knoten, deren schließender Tag ihres Elements noch nicht gelesen wurde, zusätzlich ein „unfinished“-flag, das sie als noch nicht vollständig eingelesen kenntlich macht.

Die *Architektur* der GCX Engine besteht, wie in Abbildung 3.4 dargestellt, aus den drei Komponenten: *Anfrageevaluator*, *Speichermanager* und *Dokumentprojektor*.

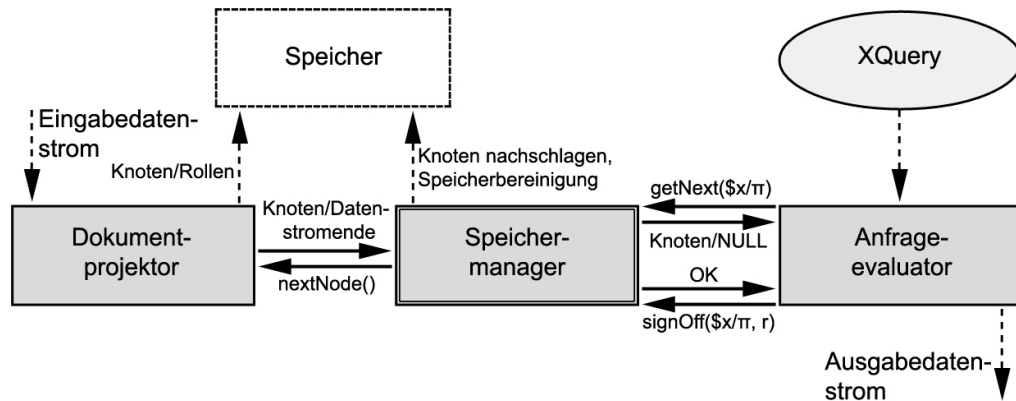


Abbildung 3.4: GCX Systemarchitektur nach [21]

Um eine strikte Trennung der einzelnen weitestgehend unabhängig voneinander operierenden Komponenten zu erhalten, erfolgt die *Kommunikation* untereinander über einfache und klar definierte *Schnittstellen (Interfaces)*. Die Kommunikation der einzelnen Komponenten ist dabei *pull-basiert*, d.h. die Komponenten werden voneinander durch *Anfragen (Requests)*, die mit einer *Antwort (Response)* bestätigt werden, „gezogen“. Eine Anfrage wird dazu vom *Anfrageevaluator* strikt sequenziell ausgewertet und die Auswertung dieser, zusammen mit der (pull-basierten) Kommunikation der einzelnen Komponenten, geht dabei wie folgt von statten:

1. Der *Anfrageevaluator* wertet die umgeschriebene Anfrage mit SignOff-Anweisungen (s. Beispiel 3.17) solange aus, bis entweder ein neuer Knoten benötigt wird (z.B. falls die Variable einer for-Klausel an den nächsten Knoten gebunden werden muss) oder eine SignOff-Anweisung auftritt. Im ersten Fall wird eine *getNext(\$x/π)*-Anfrage (Request), im zweiten Fall eine *signOff(\$x/π, r)*-Anfrage (Request) an den Speichermanager geschickt, der daraufhin die Arbeit übernimmt. Der Anfrageevaluator wartet, solange er keine Antwort (Response) vom Speichermanager erhält, mit der Fortsetzung seiner Tätigkeit bzw. mit der Fortsetzung, die Anfrage weiter auszuwerten.
2. Der *Speichermanager* sendet, falls er eine *getNext(\$x/π)*-Anfrage (Request) vom Anfrageevaluator erhält und sich dieser vom Anfrageevaluator nächst

benötigte bzw. angeforderte Knoten, der in der durch Pfad „ $\$/\pi$ “ lokalisierten Knotenmenge enthalten ist, nicht nach einem „Nachschlagen“ im Speicher befindet, eine *nextNode()*-Anfrage (Request) an den Dokumentprojektor. Das Senden von *nextNode()*-Anfragen (Requests) wird dabei von ihm solange wiederholt, bis sich entweder der vom Anfrageevaluator nächst benötigte bzw. angeforderte Knoten im Speicher befindet oder es sich ergibt, dass dieser bzw. die benötigten Daten nicht im Eingabedatenstrom vorhanden sind (z.B. nach vollständiger Abarbeitung des Eingabedatenstroms). Der Erhalt einer *signOff*($\$/\pi, r$)-Anfrage (Request) vom Anfrageevaluator wiederum veranlasst ihn, die in der Anfrage (Request) angegebene Rolle r an den bereits gespeicherten Knoten zu entfernen. Dabei verlieren alle Knoten, die durch den Pfad „ $\$/\pi$ “ lokalisiert werden, diese Rolle. In diesem Zuge wird zugleich die Speicherbereinigung von ihm durchgeführt, bei der er irrelevante Knoten nach Definition 3.14 aus dem Speicher entfernt. In Abhängigkeit der Anfrage (Request) des Anfrageevaluators antwortet der Speichermanager diesem. So sendet er die Antwort (Response) „Knoten“, d.h. einen Zeiger auf diesen, falls sich der nächst benötigte bzw. angeforderte Knoten im Speicher befindet, oder „NULL“, d.h. „keinen“ Zeiger, falls dieser nicht im Speicher vorhanden ist, nach Erhalt einer *getNext*($\$/\pi$)-Anfrage (Request) und „OK“ nach Erhalt einer *signOff*($\$/\pi, r$)-Anfrage (Request) an den Anfrageevaluator zurück.

3. Der *Dokumentprojektor*, einmal vom Speichermanager durch ein *nextNode()*-Anfrage (Request) angestoßen, verarbeitet den Eingabedatenstrom analog dem Vorgehen eines SAX-Parsers, solange bis ein Tag⁸ oder Elementtext gefunden wird, der für die Auswertung der Anfrage relevant ist. Relevant bedeutet, dass deren Element- oder Textknoten des Dokumentenbaums in der Knotenmenge bzw. den Knotenmengen, die durch die in einem oder mehreren Knoten beschriebenen Pfaden des Projektionsbaums lokalisiert werden, enthalten ist. Ist ein solcher (relevanter) Tag oder Elementtext gefunden, so wird er zusammen mit seiner zugehörigen Rolle(n) in den Speicher kopiert und der Speichermanager hierüber mit der Antwort (Response) „Knoten“ informiert.

⁸Für die Entscheidung, ob Tags für eine Anfrage bzw. eines Ausdrucks relevant sind, ist es aufgrund der zwei vorwärtsgerichteten Achsen „child“ und „descendant“ ausreichend, lediglich den öffnenden Tag zu betrachten.

Im Falle der Nichtexistenz eines relevanten Tags oder Elementtexts ist der Eingabedatenstrom zu Ende gelesen und der Speichermanager wird mit der Antwort (Response) „Datenstromende“ über dies in Kenntnis gesetzt.

Auf diese Art und Weise der Bearbeitung durch die drei Komponenten wird schrittweise sowohl eine Anfrage vom Anfrageevaluator ausgewertet als auch ein Eingabedatenstrom vom Dokumentprojektor gelesen. Dabei unterliegt die Kontrolle, wann der Eingabedatenstrom vom Dokumentprojektor gelesen werden soll, und somit auch die Speicherung neuer Knoten indirekt beim Anfrageevaluator. Die Verarbeitung des Eingabedatenstroms durch den Dokumentprojektor wird vom Anfrageevaluator über den Speichermanager nur ersucht, wenn die zur Auswertung der Anfrage bzw. eines Ausdrucks notwendigen Knoten nicht im Speicher vorhanden sind. Daraus ergibt sich, sobald eine Auswertung der Anfrage bzw. eines Ausdrucks aufgrund im Speicher befindlicher Knoten möglich ist, dass das Lesen des Eingabedatenstroms durch den Dokumentprojektor ausgesetzt wird.

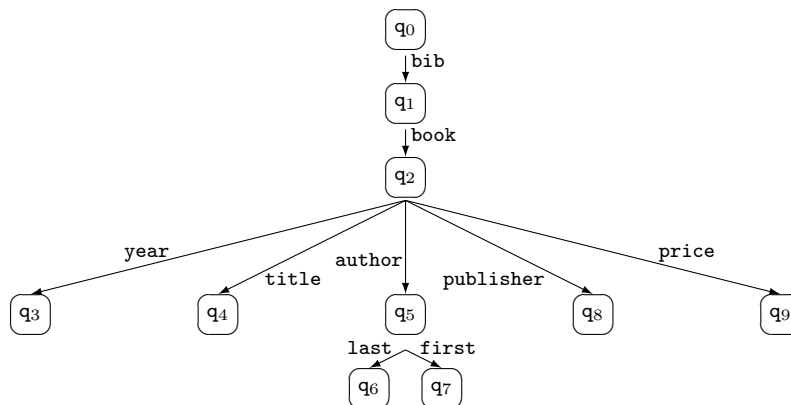
Der *Speicher*, der den bislang *projizierten Dokumentenbaum* des vom Dokumentprojektor gelesenen Teils des Eingabedatenstroms enthält, wird, um selbst einen geringen Speicherbedarf in Anspruch zu nehmen, durch einen einfachen Baum als Datenstruktur repräsentiert. Dieser besitzt zwischen den einzelnen Knoten Zeiger von Eltern auf Kinder und auf den nächsten (rechten) Geschwisterknoten und stellt somit sicher, dass Daten und somit Knoten während der Auswertung einer Anfrage nicht mehrfach gespeichert werden. Aufgrund des zugrunde liegenden Sprachfragments XQ, welches kompositionsfrei (composition-free) ist, werden alle Variablen an Knoten des sich im Speicher befindenden (projizierten) Dokumentenbaums gebunden. Des Weiteren wird mit Hilfe eines „unfinished“-flags an den gespeicherten Knoten signalisiert, ob der schließende Tag ihres Elements bereits gelesen wurde. Dieses wird, sobald der schließende Tag ihres Elements gelesen wurde, wieder entfernt.

Der *Dokumentprojektor*, der unter Verwendung des *Projektionsbaum* die Dokumentprojektion vornimmt, ist, ähnlich wie in [30] beschrieben, mit Hilfe eines „träge“ konstruierten *deterministischen endlichen Automaten (deterministic finite automaton (DFA))* umgesetzt worden. Träge bedeutet in diesem Zusammenhang nach [30], dass der DFA nach Bedarf zur Laufzeit konstruiert wird. So besteht er initial

aus nur einem Zustand (Startzustand) und wann immer ein Übergang in einen nicht existenten Zustand versucht wird, wird dieser Zustand berechnet und die Übergänge aktualisiert. So berechnet sich bspw. während der Projektion des Dokuments in Beispiel 2.1 der im nachfolgenden Beispiel 3.18 dargestellte DFA.

Beispiel 3.18 (*Deterministischer endlicher Automat für Beispiel 2.1*)

Das in Beispiel 2.1 enthaltene Dokument ergibt den nachfolgend dargestellten DFA.



Dabei besitzt ein DFA keine Übergänge zu Elementtext eines Dokuments, da die diesen zugehörigen Textknoten eines Dokumentenbaums keine Rolle erhalten.

Aufgrund der Komplexität des verwendeten bzw. konstruierten DFA und aufgrund der Tatsache, dass die Erweiterung dieses nicht Bestandteil dieser Arbeit war, wird dieser in der weiteren Arbeit als Blackbox betrachtet. Für eine genauere Beschreibung und für Beispiele des DFA wird an dieser Stelle auf [20] bzw. [21] verwiesen.

Außerdem sind zur Reduzierung der Auswertungszeiten einer Anfrage und des dazu notwendigen Speicherbedarfs die nachfolgenden *Optimierungen*, die weder von der statischen noch von der dynamischen Analyse umfasst werden, in die Implementierung der GCX Engine eingeflossen:

- „*Entfernen von redundanten Rollen eines Projektionsbaums*“

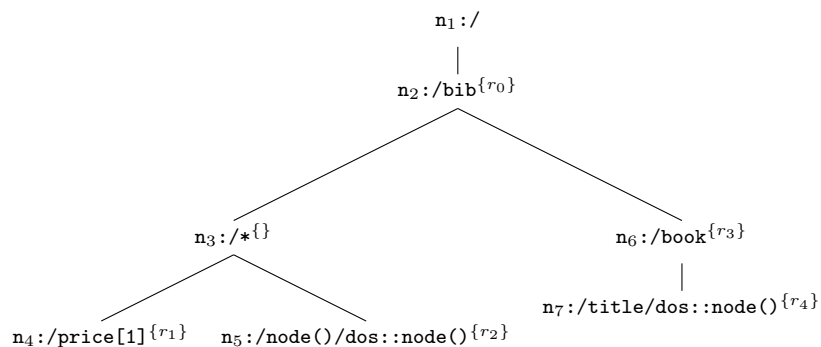
In manchen Ausnahmefällen treten redundante Rollen in Knoten eines Projektionsbaums auf, die durch Untersuchen des Projektionsbaums gefunden werden können. Diese können, wie durch das nachfolgende Beispiel 3.19 verdeutlicht wird, direkt aus dem Projektionsbaum wieder entfernt werden.

Beispiel 3.19 (Entfernen von redundanten Rollen)

Wird die Anfrage aus Beispiel 3.2 wie folgt nur leicht verändert,

```
<r> {
  for $bib in /bib return
    ((for $x in $bib/* return
      if (not(exists($x/price))) then <book> {$x/node()} </book> else ()),
    for $b in $bib/book return $b/title)
} </r>
```

so liefert sie das gleiche Resultat wie die ursprüngliche Anfrage in Beispiel 3.2 und es ergibt sich der nachfolgende zur ursprünglichen Anfrage nahezu identische Projektionsbaum,



dessen Knoten n_3 keine Rolle erhalten hat und für die auch keine SignOff-Anweisung in die Anfrage eingefügt wird.

Durch das Entfernen von redundanten Rollen in Knoten eines Projektionsbaum werden die ihren zugehörigen SignOff-Anweisungen nicht in eine Anfrage eingefügt. Dies ist auch der Grund, weshalb der Wurzelknoten des Projektionsbaums, der für die Variable $\$root$ steht, niemals eine Rolle zugewiesen bekommen hat, denn das Zuweisen und Entfernen einer Rolle für die Dokumentwurzel eines Dokumentenbaums ist immer möglich und kann somit direkt entfallen. Diese Optimierung wird in Abschnitt 6.4 nochmals aufgegriffen, weshalb an dieser Stelle auf weitere Erläuterungen verzichtet wird.

- „Zuweisen von aggregierten Rollen“

Aufgrund der Abhängigkeiten in Definition 3.5 bzw. durch das Einfügen eines Schrittes der Form „/dos::node()“ an den in einer Anfrage auftretenden Pfad werden Knoten, die ausgegeben werden, samt all ihrer Nachkommen gespeichert. Bei ihrer Speicherung erhalten (bisher) auch alle diese Knoten die gleiche Rolle zugewiesen, die wiederum (bisher) auch von all diesen Knoten

mit Hilfe einer SignOff-Anweisung entfernt werden mussten. Um die Menge der vergebenen Rollen an (diesen) zu speichernden Knoten zu reduzieren, erhält der Wurzelknoten eines auszugehenden Teilbaums eine aggregierte Rolle, die an alle seine Nachkommen „vererbt“ wird. Dies führt zu kleineren Rollenmengen an den gespeicherten Knoten und bringt somit beim Entfernen dieser Rollen nur das Entfernen dieser an einem Knoten mit sich.

- „*Hashing der gespeicherten Elementnamen*“

Anstatt die Elementnamen für Elementknoten zu speichern, werden diese durch Zahlen ersetzt. Eine bidirektionale Hash-Tabelle, die während der Dokumentprojektion gefüllt wird, bildet die Elementnamen der gespeicherten Elementknoten auf die entsprechenden Zahlen und umgekehrt ab. Somit kann nicht nur Speicherbedarf, sondern auch der Vergleich von Elementknoten, die nun im (projizierten) Dokumentenbaum durch eine Zahl repräsentiert werden, auf den Vergleich von Zahlen reduziert werden, um damit bspw. Joins zu beschleunigen.

- „*Früheres Entfernen von gespeicherten Knoten*“

Die SignOff-Anweisung, `signOff($b/title/dos::node(), r5)`, der umgeschriebenen Anfrage mit SignOff-Anweisungen in Beispiel 3.17 wird erst ausgeführt, nachdem der Titel bzw. title-Elementknoten ausgegeben wurde. Falls jedoch ein Buch mehr als einen Titel bzw. title-Elementknoten besitzen sollte, wird die Speicherbereinigung durch diese SignOff-Anweisung erst nach Ausgabe *aller* Titel bzw. title-Elementknoten ausgeführt. Um Speicherbedarf jedoch früher wieder freizugeben, werden alle Pfade in Ausgaben der Form „ $\$x/\sigma$ “ in äquivalente FR-Ausdrücke der Form „for $\$y$ in $\$x/\sigma$ return $\$y$ “ mit einer neuen Variablen $\$y$, wie nachfolgendes Beispiel 3.20 verdeutlicht, umgeschrieben.

Beispiel 3.20 (Anfrage aus Beispiel 3.2 mit früherem Entfernen)

Die Anfrage aus Beispiel 3.2 ergibt nach Anwendung der Optimierung zum „früherem Entfernen von gespeicherten Knoten“ die nachfolgend dargestellte Anfrage.

```
<r> {
  for $bib in /bib return
    ((for $x in $bib/* return
      if (not(exists($x/price))) then $x else ()),
     for $b in $bib/book return
      (for $b' in $b/title return $b'))
} </r>
```

Das Einfügen von SignOff-Anweisungen in eine Anfrage führt nun zu „for \$y in \$x/σ return (\$y, signOff(\$y, r))“ mit einer neuen Rolle *r*, wie nachfolgendes Beispiel 3.21 verdeutlicht.

Beispiel 3.21 (Anfrage aus Beispiel 3.20 mit SignOff-Anweisungen)

Die Anfrage aus Beispiel 3.20 ergibt die nachfolgend dargestellte umgeschriebene Anfrage mit SignOff-Anweisungen.

```
<r> {
  for $bib in /bib return
    ((for $x in $bib/* return
      (if (not(exists($x/price))) then $x else (),
       signOff($x, r1),
       signOff($x/price[1], r2),
       signOff($x/dos::node(), r3))),
     (for $b in $bib/book return
      (for $b' in $b/title return
       ($b',
        signOff($b', r5),
        signOff($b'/dos:node(), r6))),
       signOff($b, r4)),
     signOff($bib, r0))
} </r>
```

Auf diese Weise werden die SignOff-Anweisungen nach *jeder* Ausgabe eines Titels bzw. title-Elementknotens ausgeführt und die darin angegebene Rolle an den entsprechenden Knoten des Dokumentenbaums entfernt. Dies führt u.U. dazu, dass der gespeicherte title-Elementknoten nach jeder Ausgabe durch die nach dem Entfernen der Rolle ausgeführte Speicherbereinigung früher (wieder) aus dem Speicher entfernt wird.

Das sich in Tabelle C.1 befindende Beispiel soll das Zusammenspiel von statischer und dynamischer Analyse anhand der in Beispiel 3.2 enthaltenen Anfrage auf das sich in Beispiel 2.1 befindliche Dokument nochmals verdeutlichen.

Kapitel 4

GCX Erweiterung: Pfade mit mehreren Schritten

Basierend auf den in Kapitel 3 behandelten theoretischen Hintergründen der aktiven Speicherbereinigung und deren praktische Umsetzung in der GCX Engine, wird in diesem Kapitel die Erweiterung dieser beiden beschrieben, Anfragen zu unterstützen, die Pfade mit mehreren Schritten besitzen.

Dazu wird im nun Folgenden zunächst auf die Problematik eingegangen, die diese Erweiterung notwendig gemacht hat, und im Anschluss daran das in Abschnitt 3.1 beschriebene Sprachfragment XQ neu gefasst. Hierauf aufbauend werden im weiteren Verlauf dieses Kapitels die zum Erreichen dieser Erweiterung notwendigen Definitionen und Algorithmen eingeführt und anhand von Beispielen deren Funktion anschaulich verdeutlicht. Den Abschluss dieses Kapitels bilden die durchgeführten praktischen Veränderungen und Modifikationen zur Umsetzung dieser Erweiterung in die GCX Engine.

4.1 Pfade mit einem Schritt vs. Pfade mit mehreren Schritten

Wie bereits in Abschnitt 3.1 erwähnt, ist es durch das Sprachfragment XQ bisher nur möglich, *Pfade mit einem Schritt* in einer Anfrage zu verwenden. Dort ist die Möglichkeit erwähnt worden, Pfade von for-Klauseln, die mehrere Schritte besitzen, eventuell durch ineinander verschachtelte for-Klauseln mit Pfaden, die nur einen Schritt besitzen, wie nachfolgendes Beispiel 4.1 zeigt, umzuschreiben.

Beispiel 4.1 (Anfrage für Quelltext D.1)

Die nachfolgende Anfrage gibt die Titel aller Bücher aus.

```
<r> {  
  for $title in /bib/book/title return  
    <book> {$title} </book>  
} </r>
```

Durch Umschreiben der Pfade mit mehreren Schritten der Pfade der for-Klauseln in Pfade mit einem Schritt ergibt sich für diese die nachfolgende, zur ursprünglichen Anfrage *äquivalente*, Anfrage.

```
<r> {  
  for $bib in /bib return  
    for $book in $bib/book return  
      for $title in $book/title return  
        <book> {$title} </book>  
} </r>
```

Ein derartiges Umschreiben der Pfade von for-Klauseln mit mehreren Schritten zu Pfaden mit einem Schritt unterliegt gewissen Restriktionen. Eine dieser kann, wie nachfolgendes Beispiel 4.2 zeigt, an den in einem Schritt eines Pfades verwendeten Achsen liegen.

Beispiel 4.2 (Anfrage für Quelltext D.1)

Die nachfolgende Anfrage gibt die Kapitelnamen aller Büchern aus.

```
<r> {  
  for $book in //book return  
    <book> {  
      for $title in $book//section//title return  
        $title  
    } </book>  
} </r>
```

Durch Umschreiben der Pfade mit mehreren Schritten der Pfade der for-Klauseln in Pfade mit einem Schritt ergibt sich für diese die nachfolgende, zur ursprünglichen Anfrage *nicht äquivalente*, Anfrage.

```
<r> {  
  for $book in //book return  
    <book> {  
      for $section in $book//section return  
        for $title in $section//title return  
          $title  
    } </book>  
} </r>
```

Im Gegensatz zu Beispiel 4.1 besteht in Beispiel 4.2 das Problem der Umschreibung der Pfade der for-Klauseln an der verwendeten Achse „descendant“ der einzelnen

Schritte. Die (einzelnen) Schritte eines Pfades lokalisieren, wie in Abschnitt 2.3 beschrieben, eine Menge von Knoten, die die Kontextknoten des nachfolgenden Schrittes sind. Dabei muss natürlich für einen Pfad beachtet werden, dass es u.U. aus Sicht mehrerer Kontextknoten eines vorhergehenden Schrittes zu Überschneidungen der lokalisierten Knoten des diesen nachfolgenden Schrittes kommen kann, d.h. dieser nachfolgende Schritt lokalisiert durch seine Achse und Knotentest aus Sicht mehrere Kontextknoten denselben Knoten eines Dokumentenbaums. Dieses Szenario, dass derselbe Knoten von mehreren Kontextknoten durch einen nachfolgenden Schritt lokalisiert wird, kann bspw. durch die Verwendung der Achse „descendant“ und gleichem oder ineinander enthaltenem Knotentest in zweien oder mehreren Schritten eines Pfades auftreten.

Bei Pfaden mit mehreren Schritten sind die einzelnen durch einen Schritt dieses Pfades lokalisierten Knotenmengen für sich genommen ohne Duplikate, d.h. Knoten, die bereits aus Sicht eines Kontextknoten lokalisiert werden, treten nur einmalig in dieser Knotenmenge auf. Dies ist durch das Umschreiben der Pfade von for-Klauseln, so dass diese nur einen Schritt besitzen, u.U. nicht immer gewährleistet. Durch das Umschreiben zu Pfaden mit einem Schritt wird eine Knotenmenge lokalisiert, ohne Kontextknoten eines vorherigen Schrittes betrachten zu müssen. Damit ist diese lokalisierte Knotenmenge aufgrund keines vorherigen Schrittes stets ohne Duplikate. Daraus folgt, dass das erhaltene Resultat einer Anfrage mit Pfaden, die mehrere Schritte besitzen, im Vergleich zum Resultat einer anscheinend zu dieser Anfrage äquivalenten Anfrage, mit Pfaden, die nur einen Schritt besitzen, nicht identisch ist. Der Grund hierfür liegt an den durch die Pfade mit einem Schritt größeren (lokalisierten) Knotenmengen, weshalb mehr Ausgaben erfolgen bzw. die Iteration von for-Klauseln öfter durchlaufen wird als durch die Verwendung von Pfaden mit mehreren Schritten.

4.2 Sprachfragment XQ'

Das in der nachfolgenden Abbildung 4.1 dargestellte Sprachfragment stellt das aus dem ursprünglichen Sprachfragment XQ aus Abschnitt 3.1 neu erhaltene Sprachfragment XQ' dar.

$$\begin{aligned}
 XQuery & ::= \epsilon \mid XMLExpr \\
 XMLExpr & ::= \langle QName \rangle NestedXMLExpr \langle / QName \rangle \\
 & \quad \mid \langle QName \rangle \langle / QName \rangle \mid \langle QName / \rangle \\
 NestedXMLExpr & ::= \{ QExpr \} \mid String \mid XMLExpr \mid NestedXMLExpr NestedXMLExpr \\
 QExpr & ::= ReturnQExpr \mid QExpr, QExpr \\
 ReturnQExpr & ::= QExprSingle \mid (QExpr) \mid () \\
 QExprSingle & ::= \text{“String”} \mid FWRExpr \mid IfExpr \mid VarExpr \mid NestedXMLExpr \\
 FWRExpr & ::= ForClause [\textbf{where} Condition]? \textbf{return} ReturnQExpr \\
 ForClause & ::= \textbf{for} \$ VarName \textbf{in} VarAxisExpr [, \$ VarName \textbf{in} VarAxisExpr]^* \\
 IfExpr & ::= \textbf{if} (Condition) \textbf{then} ReturnQExpr \textbf{else} ReturnQExpr \\
 Condition & ::= \textbf{fn:true}() \mid \textbf{fn:false}() \mid \textbf{fn:exists}(VarAxisExpr) \\
 & \quad \mid (Condition) \mid Operand RelOp Operand \mid \textbf{fn:not}(Condition) \\
 & \quad \mid Condition \textbf{and} Condition \mid Condition \textbf{or} Condition \\
 Operand & ::= VarExpr \mid \text{“String”} \\
 RelOp & ::= < \mid <= \mid >= \mid > \mid = \mid != \\
 VarExpr & ::= \$ VarName \mid VarAxisExpr \\
 VarAxisExpr & ::= \$ VarName PathExpr \mid PathExpr \\
 PathExpr & ::= / \mid PathStepExpr \\
 PathStepExpr & ::= Axis NodeTest \mid Axis NodeTest / PathStepExpr \\
 Axis & ::= / \mid // \mid \textbf{child::} \mid // \mid \textbf{descendant::} \\
 NodeTest & ::= \textbf{node}() \mid \textbf{text}() \mid * \mid QName
 \end{aligned}$$

QName := Elementname
String := Zeichenkette
VarName := XQuery Variable (z.B. $\$x$, $\$y$, ...)

Abbildung 4.1: XQuery Sprachfragment XQ'

Dieses Sprachfragment XQ' wird in der weiteren Arbeit, da es das ursprüngliche Sprachfragment XQ' aus Abschnitt 3.1 enthält und nun die Erweiterung um Pfade mit mehreren Schritten umfasst, als Sprachfragment XQ bezeichnet.

Die Veränderungen zur Erweiterung des Sprachfragments XQ zur Unterstützung von Pfaden mit mehreren Schritten in einer Anfrage sind in Abbildung 4.1 ab Ausdruck „VarAxisExpr“ zu erkennen. Die Verwendung von Pfaden in einer Anfrage ist nun nicht mehr beschränkt, d.h. es können nun Pfade mit beliebiger Anzahl von Schritten durch das Sprachfragment XQ gebildet werden.

4.3 Theoretische Grundlagen und Hintergründe

Für die zur Umsetzung dieser Erweiterung notwendigen theoretischen Grundlagen und Maßnahmen wird die nachfolgende Anfrage aus Beispiel 4.3 betrachtet, die Pfade mit mehreren Schritten enthält. Sie bildet dabei auch die Grundlage einiger noch folgender Beispiele.

Beispiel 4.3 (*Anfrage für Quelltext D.1*)

Die nachfolgende Anfrage gibt alle Buchtitel zusammen mit den Namen der Kapitel auf der ersten Ebene, falls solche existieren, sonst den Tag `<no-subsections/>` aus.

```
<r> {
  for $book in /bib/book return
    <book>
    <title> {$book/title/text()} </title>
    <sections> {
      for $chapter in $book/toc/chapter return
        (
          if (exists($chapter/section/title)) then
            for $section in $chapter/section/title return
              <subsection><title> {$section/text()} </title></subsection>
          else(<no-subsections/>)
        )
    } </sections>
  </book>
}</r>
```

Die erste Veränderung ist für die in Definition 3.5 genannten Abhängigkeiten vorgenommen worden. Dazu wird die nachfolgende Definition 4.1 benötigt, die den Präfix eines Pfades definiert.

Definition 4.1 (Präfixpfad π_i)

Sei P ein relativer Pfad bestehend aus $n \geq 0$ Schritten und P_i mit $0 \leq i \leq n$ der Pfad P bestehend aus den ersten i Schritten. Der *Präfixpfad* eines Pfades P , bezeichnet mit π_i und $i \leq n$, ist wie folgt definiert

$$\pi_i \stackrel{def}{=} \begin{cases} \epsilon, & \text{falls } i = 0 \\ P_i, & \text{sonst.} \end{cases}$$

Beispiel 4.4 (Präfixpfad π_i für verschiedene Pfade P)

Sei $P_1 = bib//title$ dann ist für $n = 2$ Schritte $\pi_1 = bib$ und $\pi_2 = P_1 = bib//title$.

Sei $P_2 = bib/book/title$, dann gilt für $\pi_{n-1}/title$

mit $n = 3$ Schritten $\pi_2 = bib/book$.

Sei $P_3 = (/)$ dann ist für $n = 0$ Schritte $\pi_0 = \epsilon$.

Sei $\beta = \text{„for } \$y \text{ in } \$x/bib/book//title \text{ return } \alpha\text{“}$, dann gilt für $\$x/\pi_n$

mit $n = 3$ Schritten $\pi_1 = bib$, $\pi_2 = bib/book$ und $\pi_3 = bib/book//title$ und

für $\$x/\pi_{n-1}/title$ gilt $\pi_2 = bib/book$.

Sei $\beta = \text{„for } \$y \text{ in } \$x//title \text{ return } \alpha\text{“}$, dann gilt für $\$x/\pi_{n-1}/title$

mit $n = 1$ Schritten $\pi_0 = \epsilon$.

Sei $\beta = \text{„for } \$y \text{ in } \$x \text{ return } \alpha\text{“}$, dann gilt für $\$x/\pi_n$

mit $n = 0$ Schritten $\pi_0 = \epsilon$.

Definition 4.2 (Abhängigkeiten $dep'_Q(\$x)$)

Sei Q eine Anfrage in XQ und $\$x \in Vars_Q$. Die Menge der *Abhängigkeiten der Variablen* $\$x$, bezeichnet mit $dep'_Q(\$x)$, sind wie folgt definiert. Sei $\beta \preceq Q$ mit Rollenfunktion $r_Q(\beta) = r$ und π_i ein Präfixpfad, dann:

- $\langle \pi_{n-1}/axis::\nu[1], r \rangle \in dep'_Q(\$x)$, falls $\beta = \text{„exists}(\$x/\pi_{n-1}/axis::\nu)\text{“}$,
- $\langle \pi_{n-1}/axis::\nu/dos::node(), r \rangle \in dep'_Q(\$x)$, falls β entweder ein Ausdruck in einer Ausgabe der Form $\text{„}\$x/\pi_{n-1}/axis::\nu\text{“}$, ein bedingter Ausdruck der Form $\text{„}\$x/\pi_{n-1}/axis::\nu \text{ RelOp } \chi\text{“}$ oder $\text{„}\chi \text{ RelOp } \$x/\pi_{n-1}/axis::\nu\text{“}$, und
- $\langle dos::node(), r \rangle \in dep'_Q(\$x)$, falls β ein Ausdruck in einer Ausgabe der Form $\text{„}\$x\text{“}$.

Da die in Definition 4.2 genannten Abhängigkeiten, die Abhängigkeiten aus Definition 3.5 inkludieren, werden diese in der weiteren Arbeit als $dep_Q(\$x)$ bezeichnet. Somit ergeben sich die im nachfolgenden Beispiel 4.5 dargestellten Abhängigkeiten für die Anfrage aus Beispiel 4.3.

Beispiel 4.5 (Abhängigkeiten $dep_Q(\$x)$ für Beispiel 4.3)

Für die Anfrage aus Beispiel 4.3 ergeben sich die nachfolgenden Abhängigkeiten.

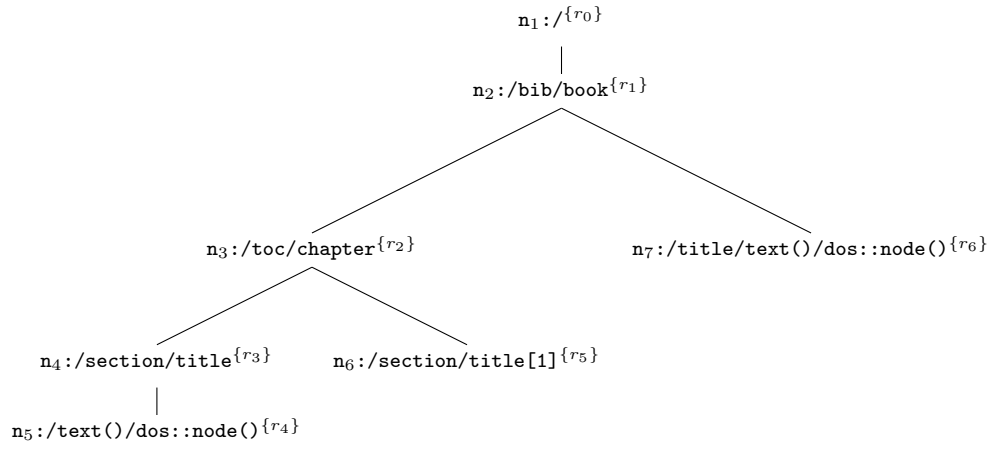
$$\begin{aligned} dep_Q(\$book) &= \{\langle child::title/child::text()/dos::node(), r_6 \rangle\} \\ dep_Q(\$chapter) &= \{\langle child::section/child::title[1], r_5 \rangle\} \\ dep_Q(\$section) &= \{\langle child::text()/dos::node(), r_4 \rangle\} \end{aligned}$$

Die in der Menge $Vars_Q$ ebenfalls enthaltene Variable $\$root$ besitzt hingegen keine Abhängigkeiten.

Neben der Veränderung der Abhängigkeiten haben sich auch die Herleitung eines Projektionsbaums und auch der Erhalt der den zu speichernden Knoten zu vergebenden Rollen geändert. Die Herleitung eines einer Anfrage zugehörigen Projektionsbaums erfolgt weiterhin durch die Verwendung des einer Anfrage zugehörigen Variablenbaums. Nach der Erstellung des Variablenbaums aus einer Anfrage wird nun zunächst die Wurzel des Variablenbaums mit „/“ beschriftet, mit dem analog zur bisherigen Herleitung eines Projektionsbaums angedeutet wird, dass alle Pfade, die durch einen Knoten des Projektionsbaums beschrieben werden, absolut sind. Als nächstes werden nun alle mit einer Variablen, $\$x$, benannten Knoten n des Variablenbaums mit dem Pfad seiner zugehörigen for-Klausel $\beta = \text{„for } \$x \text{ in } \$y/\pi_n \text{ return } \alpha\text{“}$, über dessen Knotenmenge diese iteriert, ersetzt, d.h. die Variable $\$x$ wird durch „/ π_n “ ersetzt. Als nächstes werden für jede Variable $\$x$ und für jede Abhängigkeit $\langle \$x/\pi_n \rangle \in dep_Q(\$x)$ ein Knoten n mit Beschriftung „/ π_n “ und eine Kante von $\$x$ nach n in den Variablenbaum eingefügt. Nachdem dies erfolgt ist und der Variablenbaum nun nur noch Knoten, die mit Pfaden beschriftet sind, besitzt, werden durch einen Tiefendurchlauf des Variablenbaums die Rollen den Knoten vergeben und man erhält, wie im nachfolgenden Beispiel 4.6 dargestellt, den einer Anfrage zugehörigen Projektionsbaum.

Beispiel 4.6 (Projektionsbaum mit Rollen für Beispiel 4.3)

Für die Anfrage aus Beispiel 4.3 ergibt sich der nachfolgend dargestellte Projektionsbaum mit Rollen.



Jeder Knoten n eines Projektionsbaums beschreibt einen Pfad, der sich aus der Konkatenation der einzelnen Pfade von der Wurzel „/“ des Projektionsbaums zu diesem Knoten n ergibt. Aufgrund der nun auftretenden Pfade der einzelnen Knoten eines Projektionsbaums sind daher einige Besonderheiten bei der Speicherung von Knoten eines Dokumentenbaums zu beachten. Nur Knoten, die durch einen beschriebenen Pfad lokalisiert werden, erhalten eine Rolle zugewiesen, die aus dem Knoten des Projektionsbaums stammt, aus dem der beschriebene Pfad gewonnen wird. Knoten wiederum, die die Kontextknoten der einzelnen Schritte eines Pfades darstellen, werden ebenfalls zur Navigation in dem im Speicher befindlichen (projizierten) Dokumentenbaum gespeichert, erhalten allerdings keine Rolle zugewiesen. Dies stellt einen großen Unterschied zu der Speicherung bzw. Zuweisung von Rollen der in Kapitel 3 beschriebenen aktiven Speicherbereinigung dar.

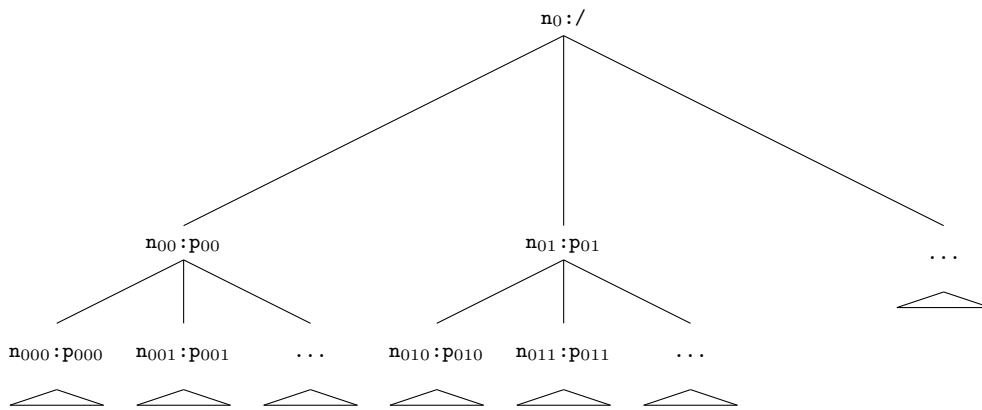
Eine weitere Veränderung stellt die Vergabe einer Rolle der Wurzel eines Projektionsbaums dar. Diese Rolle, die der Dokumentwurzel eines Dokumentenbaums vergeben wird, ist dieser in Kapitel 3 nicht vergeben worden.

Bei der Zuweisung von Rollen den zu speichernden Knoten eines Dokumentenbaums ist es möglich, dass diese Knoten zum einen mehrere unterschiedliche Rollen erhalten, zum anderen mehrmals dieselbe Rolle zugewiesen bekommen. Das liegt an den auszuführenden SignOff-Anweisungen in einer Anfrage, die durch die in ihnen enthaltenen Pfade den gleichen Knoten mehrmals bei der Iteration einer for-Klausel

lokalisieren können. Aus diesem Grund müssen diesen Knoten eines Dokumentenbaums bei ihrer Speicherung die Rollen der Knoten des Projektionsbaums, die diese durch ihre beschriebenen Pfade lokalisieren, mehrfach zugewiesen werden. Dazu wird die nachfolgende Definition 4.3 benötigt, die zunächst die Knotenmenge eines Projektionsbaums definiert, die beim Speichern eines Knotens eines Dokumentenbaums für die Zuweisung einer Rolle verantwortlich ist.

Definition 4.3 (Knotenmenge eines Projektionsbaums)

Sei pt ein Projektionsbaum der Form:



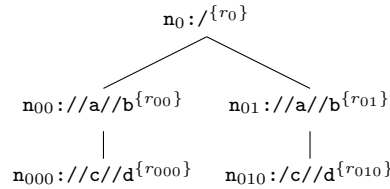
Jeder Knoten $n_{i_1...i_n}$ in pt beschreibt einen Pfad $/p_{i_1}/p_{i_1i_2}/\dots/p_{i_1...i_n}$, bezeichnet mit $path(n_{i_1...i_n})$, von der Wurzel von pt , n_0 , zu diesem Knoten. Sei weiter $r_{i_1...i_n}$ die Rolle des Knotens $n_{i_1...i_n}$ und $\langle t_i \rangle$ der gerade gelesene Tag (oder Elementtext) des als Datenstrom erhaltenen Dokuments, der den derzeitigen Pfad im Dokumentenbaum zu



komplettiert. Die *Knotenmenge* eines Projektionsbaum ist definiert als die Menge $N_{t_0...t_i} = \{n_{i_1...i_n} \in pt \mid path(n_{i_1...i_n}) \text{ selektiert } \langle t_i \rangle\}$.

Beispiel 4.7 (Knotenmenge eines Projektionsbaums)

Sei der nachfolgende Projektionsbaums betrachtet



und das nachfolgende Dokument als Datenstrom erhalten

$\langle a \rangle \langle a \rangle \langle b \rangle \langle b \rangle \langle c \rangle \langle c \rangle \langle d \rangle \dots$

und sei $\langle t_0 \rangle = \langle a \rangle$ und $\langle t_i \rangle = \langle d \rangle$, dann selektiert $path(n_{000}) = //a//b//c//d$ und $path(n_{010}) = //a//b/c//d \langle t_0 \rangle \dots \langle t_i \rangle$, d.h. $N_{t_0 \dots t_i} = \{n_{000}, n_{010}\}$.

Die Anzahl zugewiesener Rollen eines zu speichernden Knotens eines Dokumentenbaums wird mit Hilfe der nachfolgenden Definition 4.4 festgelegt. Die Berechnung der Anzahl zu erhaltener Rollen eines zu speichernden Knotens eines Dokumentenbaums erfolgt allgemein zunächst durch die Bestimmung der Knoten im Projektionsbaum, die diesen Knoten durch ihre beschriebenen Pfade lokalisieren. Anschließend wird von diesen Knoten die erste direkte Vorgängervariable (fsa) verwendet, um den den ersten direkten Vorgängervariablen repräsentierenden Knoten im Projektionsbaum zu bestimmen. Die durch diese repräsentierenden Knoten beschriebenen Pfade werden nun dazu genutzt, um mit Hilfe der in Definition 4.4 genannten Anfrage die zu erhaltene Anzahl von Rollen zu bestimmen. Aufgrund der Komplexität der Berechnung sind mehrere unterschiedliche Beispiele, Beispiel 4.8, Beispiel 4.9 und Beispiel 4.10, nachfolgend aufgeführt.

Definition 4.4 (Anzahl Rollen eines zu speichernden Knotens)

Sei Q eine Anfrage in XQ, $\$x_{i_1\dots i_n} \in \text{Vars}_Q$ und pt ein Projektionsbaum. Sei weiter $n_{i_1\dots i_n}$ die Knoten von pt und $r_{i_1\dots i_n}$ die Rollen der Knoten $n_{i_1\dots i_n}$. Sei weiter $\$x_{i_1\dots i_n}$ die zu $n_{i_1\dots i_n}$ zugehörigen (ehemaligen) Variablen des Variablenbaums und $m_{i_1\dots i_m} = \text{fsa}_Q(\$x_{i_1\dots i_n})$ die ersten direkten Vorgängervariablen (fsa) von diesen bzw. deren repräsentierenden Knoten in pt . Sei weiter $p_{i_1\dots i_m}$ die Pfade der Knoten $m_{i_1\dots i_m}$ und $\langle t_i \rangle$ der gerade gelesene Tag (oder Elementtext) des als Datenstrom erhaltenen Dokuments, der den derzeitigen Pfad im Dokumentenbaum komplettiert. Dann erhält $\langle t_i \rangle$ bei seiner Speicherung die Rollen $r_{i_1\dots i_n}$ für jeden Knoten $n_{i_1\dots i_n}$ aus dem Resultat der nachfolgenden Anfrage

```
let $xml := <t1><t2>...<t_i/>...</t2></t1> return
fn:count(
  for $i1 in $xml return
    for $i2 in $i1/p_i2 return
      for $i3 in $i2/p_i3 return
        ...
          for $i_m in $i_{m-1}/p_{i_m} return
            <one/>
)
```

anzahlmäßig zugewiesen.

Beispiel 4.8 (Anzahl Rollen eines zu speichernden Knotens)

Für den Projektionsbaum aus Beispiel 4.7 und das nachfolgende Dokument
`<a><a><c><c><d/></c></c>`

ergibt sich für den zu speichernden Knoten des Elements `<d/>` aufgrund des beschriebenen Pfades des Knotens n_{000} des Projektionsbaums die nachfolgende Anfrage

```
let $xml := <a><a><b><b><c><c><d/></c></c></b></b></a></a> return
fn:count(
  for $i1 in $xml return
    for $i2 in $i1//a//b return
      for $i3 in $i2//c//d return
        <one/>
)
```

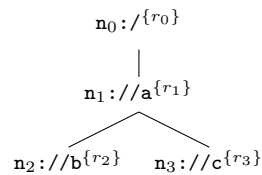
und er erhält dabei *zweimal* die Rolle r_{000} und analog (aufgrund des beschriebenen Pfades des Knotens n_{010} des Projektionsbaums) *einmal* die Rolle r_{010} zugewiesen.

Beispiel 4.9 (Anzahl Rollen eines zu speichernden Knotens)

Aus der nachfolgenden Anfrage

```
<r> {
  for $a in //a return
    for $b in $a//b return
      for $c in $a//c return
        <c/>
} </r>
```

ergibt sich der nachfolgende Projektionsbaum.



Diese Anfrage auf das nachfolgende Dokument ausgewertet

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

ergibt für den zu speichernden Knoten des zweiten Elements/Tags `` aufgrund des beschriebenen Pfades des Knotens n_2 des Projektionsbaums die nachfolgende

Anfrage

```
let $xml := <a><a><b><b><c/></b></b></a></a> return
fn:count(
  for $i1 in $xml return
    for $i2 in $i1//a return
      for $i3 in $i2//b return
        <one/>
)
```

und er erhält dabei *zweimal* die Rolle r_2 zugewiesen. Analog ergibt sich für den zu speichernden Knoten des Elements `<c/>` aufgrund des beschriebenen Pfades des Knotens n_3 des Projektionsbaums die nachfolgende Anfrage

```
let $xml := <a><a><b><b><c/></b></b></a></a> return
fn:count(
  for $i1 in $xml return
    for $i2 in $i1//a return
      <one/>
)
```

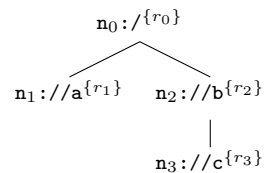
und er erhält dabei *zweimal* die Rolle r_3 zugewiesen.

Beispiel 4.10 (Anzahl Rollen eines zu speichernden Knotens)

Aus der nachfolgenden Anfrage

```
<r> {
  for $a in //a return
    for $b in //b return
      for $c in $b//c return
        <c/>
} </r>
```

ergibt sich der nachfolgende Projektionsbaum.



Diese Anfrage auf das nachfolgende Dokument ausgewertet

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

ergibt für den zu speichernden Knoten des Elements `<c/>` aufgrund des beschriebenen Pfades des Knotens n_3 des Projektionsbaums die nachfolgende Anfrage

```
let $xml := <a><a><b><b><c/></b></b></a></a> return
fn:count(
  for $i1 in $xml return
    <one/>
)
```

und er erhält dabei *einmal* die Rolle r_3 zugewiesen.

4.4 Implementierungen

Die praktische Umsetzung der Erweiterung zur Unterstützung von Pfaden mit mehreren Schritten in einer Anfrage hat *tiefgreifende Veränderungen* in der (ursprünglichen) Implementierung des Ausgangspunktes erforderlich gemacht. Diese haben sich über alle drei Komponenten – *Anfrageevaluator*, *Speichermanager* und *Dokumentprojektor* – der Architektur der GCX Engine erstreckt. An dieser Stelle werden nur die wesentlichen Veränderungen und Modifikationen genannt.

Der *Anfrageevaluator* hat bisher nur Pfade mit einem Schritt akzeptiert bzw. verarbeiten können, weshalb er für diese Erweiterung an verschiedenen Stellen umgeschrieben und verändert werden musste.

Der *Speichermanager*, der zum einen für das „Nachschlagen“ im Speicher der durch einen Pfad lokalisierten Knoten zuständig ist und zum anderen die Speicher-

bereinigung ausführt, ist vollständig reimplementiert worden. Dieser hat ursprünglich nur Knoten, die durch einen Pfad mit einem Schritt lokalisiert werden, im Speicher „nachschiessen“ können. Die Veränderungen der Speicherbereinigung sind erfolgt, da einerseits Veränderungen an der Ausführung der SignOff-Anweisungen vorgenommen wurden, andererseits Veränderungen des Rollenmanagements erfolgten.

Der *Dokumentprojektor* ist ebenfalls nahezu vollständig reimplementiert worden, da der für die Dokumentprojektion zuständige DFA ebenfalls die Bearbeitung von Pfaden mit mehreren Schritten nicht vollziehen konnte. Die vom Dokumentprojektor bzw. DFA durchgeführte Dokumentprojektion ist dahingehend reimplementiert worden, dass auch die Vorfahren – falls diese nicht benötigt werden – von relevanten Knoten entfernt werden. Dieses Verwerfen von nicht benötigten Vorfahren war in der (ursprünglichen) Implementierung des Ausgangspunktes nicht mehr vorhanden. In diesem Zuge ist die Zuweisung von Rollen auch auf Textknoten erweitert worden, so dass nun alle Knotentypen, Elementknoten und Textknoten, des sich im Speicher befindlichen Dokumentenbaums eine Rolle zugewiesen bekommen können. Auch erhält jetzt die Dokumentwurzel des sich im Speicher befindenden Dokumentenbaums stets eine Rolle zugewiesen, die sich aus den notwendigen Veränderungen des Projektionsbaums ergeben haben. Der Projektionsbaum hat dazu eine Rolle seiner Wurzel zugewiesen bekommen, die stets der Dokumentwurzel vergeben wird. Bei diesen Veränderungen und Modifikationen ist auch der im Ausblick aus [20] bzw. [21] beschriebene „*Rollenzähler*“ umgesetzt worden. Dadurch wird den zu speichernden Knoten eines Dokumentenbaums – nicht wie ursprünglich – eine Rollenmenge vergeben, sondern lediglich ein Zähler für jeden Knoten unterhalten, der die Anzahl ihm vergebener Rollen zählt. Dieser „Rollenzähler“ setzt nun das Prinzip des *Zählens von Referenzen* (*reference counting*) vollständig um. Weiter erhielt der Dokumentprojektor eine Komponente zur Prüfung der *Wohlgeformtheit* eines Dokuments. Diese prüft einige der in Abschnitt 2.1 genannten Bedingungen zur Wohlgeformtheit eines Dokuments. Diese Komponente ist an ein „Deaktivierungs“-flag¹ gebunden worden, so dass diese, falls eine Prüfung der Wohlgeformtheit eines Dokuments nicht erfolgen soll, ausgeschaltet werden kann. Standardmäßig ist die-

¹Das „Deaktivierungs“-flag zum Ausschalten der Prüfung der Wohlgeformtheit eines Dokuments ist dabei in Form eines Compiler-flags erfolgt.

se deaktiviert und kommt innerhalb der gesamten weiteren Arbeit (inklusive der durchgeführten Experimente) nicht zum Tragen. Das Einfügen dieser Komponente hat es allerdings erforderlich gemacht, den vom Dokumentprojektor zum Einlesen des als Datenstrom erhaltenen Dokuments verwendeten *SAX-Parser* anzupassen. Des Weiteren ist die Speicherung nur des ersten Knotens, der durch den in der Funktion „exists“ stehenden Pfad lokalisiert wird, entfernt worden. Die Idee allerdings, nur den ersten Knoten für diese Funktion zu speichern, wird in der Theorie der aktiven Speicherbereinigung weiter geführt.

Kapitel 5

GCX Erweiterung: Aggregatfunktionen

In diesem Kapitel wird auf die im Rahmen dieser Arbeit hinzugefügten *Aggregatfunktionen* für die *aktive Speicherbereinigung* und deren praktische Umsetzung in der GCX Engine eingegangen. Dabei bestand das Ziel nicht darin, die Aggregatfunktionen schlicht in das vorhandene Framework zu implementieren, sondern vielmehr diese darin *einzubetten*, um sie mit der Idee der aktiven Speicherbereinigung bzw. der der GCX Engine in Einklang zu bringen. Aus diesem Grund sollten sie in Bezug auf die zur Auswertung einer Anfrage benötigten Zeit und den dazu notwendigen Speicherbedarf möglichst *effizient* umgesetzt werden.

Im nun Folgenden wird zunächst das *Sprachfragment XQ* neu definiert, so dass es auch Aggregatfunktionen umfasst, und sowohl dieses als auch die weiteren *Veränderungen*, die im Rahmen der Erweiterung der Aggregatfunktionen erfolgt sind, beschrieben. Schließlich wird die zugrunde liegende *Idee* zur *Einbettung* der Aggregatfunktionen erörtert und die notwendig werdenden Änderungen der *Abhängigkeiten* von Definition 4.2 vorgenommen.

5.1 Sprachfragment XQ''

Die nachfolgende Abbildung 5.1 stellt das aus dem ursprünglichen Sprachfragment XQ aus Abschnitt 4.2 neu erhaltene Sprachfragment XQ'' dar.

$$\begin{aligned}
 XQuery & ::= \epsilon \mid XMLExpr \\
 XMLExpr & ::= \langle QName \rangle NestedXMLExpr \langle / QName \rangle \\
 & \quad \mid \langle QName \rangle \langle / QName \rangle \mid \langle QName / \rangle \\
 NestedXMLExpr & ::= \{ QExpr \} \mid String \mid XMLExpr \mid NestedXMLExpr NestedXMLExpr \\
 QExpr & ::= ReturnQExpr \mid QExpr, QExpr \\
 ReturnQExpr & ::= QExprSingle \mid (QExpr) \mid () \\
 QExprSingle & ::= "String" \mid FWRExpr \mid IfExpr \mid VarExpr \mid AggregateFunct \\
 & \quad \mid NestedXMLExpr \\
 FWRExpr & ::= ForClause [**where** Condition]? **return** ReturnQExpr \\
 ForClause & ::= **for** $ VarName **in** VarExpr [, $ VarName **in** VarExpr]* \\
 IfExpr & ::= **if** (Condition) **then** ReturnQExpr **else** ReturnQExpr \\
 Condition & ::= **fn:true()** \mid **fn:false()** \mid **fn:exists**(VarExpr) \\
 & \quad \mid (Condition) \mid Operand RelOp Operand \mid **fn:not**(Condition) \\
 & \quad \mid Condition **and** Condition \mid Condition **or** Condition \\
 Operand & ::= VarExpr \mid AggregateFunct \mid "String" \\
 RelOp & ::= < \mid <= \mid >= \mid > \mid = \mid != \\
 AggregateFunct & ::= **fn:sum**(VarExpr) \mid **fn:avg**(VarExpr) \\
 & \quad \mid **fn:min**(VarExpr) \mid **fn:max**(VarExpr) \\
 & \quad \mid **fn:count**(VarExpr) \mid **fn:stddev_samp**(VarExpr) \\
 & \quad \mid **fn:stddev_pop**(VarExpr) \mid **fn:var_samp**(VarExpr) \\
 & \quad \mid **fn:var_pop**(VarExpr) \mid **fn:list**(VarExpr) \\
 VarExpr & ::= $ VarName \mid VarAxisExpr \\
 VarAxisExpr & ::= $ VarName PathExpr \mid PathExpr \\
 PathExpr & ::= / \mid PathStepExpr \\
 PathStepExpr & ::= Axis NodeTest \mid Axis NodeTest / PathStepExpr \\
 Axis & ::= / \mid // \mid /**child::** \mid // \mid /**descendant::** \\
 NodeTest & ::= **node()** \mid **text()** \mid * \mid QName
 \end{aligned}$$

QName := Elementname

String := Zeichenkette

VarName := XQuery Variable (z.B. \$x, \$y, ...)

Abbildung 5.1: XQuery Sprachfragment XQ"

Dieses *Sprachfragment XQ*” wird, da es das ursprüngliche Sprachfragment XQ aus Abschnitt 4.2 inkludiert und nun die Erweiterung um Aggregatfunktionen umfasst, in der weiteren Arbeit als *Sprachfragment XQ* bezeichnet.

Die Verwendung von Aggregatfunktionen kann dabei sowohl in *Ausgaben* als auch in *bedingten Ausdrücken* erfolgen und ihr *Parameter* ist entweder eine *Variable*, die durch eine vorherige *for*-Klausel definiert wird, oder ein *Pfad*.¹

In der weiteren Arbeit wird, wie bereits bei anderen Funktionen des Sprachfragments XQ, auch bei Aggregatfunktionen auf das Präfix „fn:“ verzichtet.

Alle hinzugefügten Aggregatfunktionen, mit Ausnahme von „*min*“, „*max*“, „*count*“ und „*list*“, berechnen ihr jeweiliges Ergebnis aus (reinen) *Zahlenwerten*². Diese stammen aus den als Parameter übergebenen Pfad lokalisierten (Inhalten der) *Textknoten* eines Dokumentenbaums. Da die Inhalte von Textknoten eines Dokumentenbaums stets alphanumerische Zeichenketten darstellen, wird versucht, diese in der GCX Engine in einen (reinen) Zahlenwert zu überführen und in das Ergebnis einer Aggregatfunktion hinzuzurechnen. Die einzelnen in die Berechnung des Ergebnisses dieser Aggregatfunktionen einfließenden (reinen) Zahlenwerte werden dabei, wie nachfolgendes Beispiel 5.1 zeigt, für den Fall, dass die durch einen Pfad lokalisierten Knoten keine Textknoten darstellen, durch die *Konkatenation* der einzelnen (Zahlen-)Werte aller nach einem lokalisierten Knoten befindlichen (Inhalte der) Textknoten eines Dokumentenbaums zu einem einzigen (Zahlen-)Wert zusammengefasst, bevor versucht wird, diesen für eine Aggregatfunktion in einen (reinen) Zahlenwert zu überführen.

¹Im Weiteren wird angenommen, dass der Parameter einer Aggregatfunktion stets ein Pfad und keine Variable ist. Diese Annahme stellt keine Einschränkung für die in diesem Kapitel nachfolgenden Sachverhalte dar und diese können daher auch auf eine Variable als Parameter einer Aggregatfunktion übertragen werden.

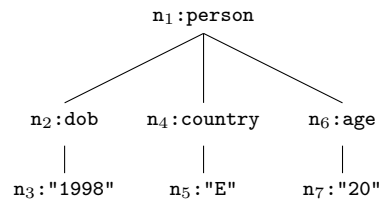
²Die einzelnen Zahlenwerte zur Berechnung des Ergebnisses einer Aggregatfunktion sind dabei gemäß des XPath- und XQuery-Standards in [38] vom Datentyp „xs:double“, welcher nach [35] einen Wertebereich aus Werten der Form $m \times 2^e$ besitzt, wobei die Mantisse m vom Datentyp Integer mit einem Absolutwert kleiner 2^{53} und der Exponent e eine Zahl ist, die mit eingeschlossenen Grenzen zwischen -1075 und 970 liegt. Weiter beinhaltet der Datentyp „xs:double“ die drei speziellen Werte positiv und negativ unendlich („INF“, „-INF“) und not-a-number („NaN“). Daher stellen bspw. die alphanumerischen Zeichenketten $-1E3$, $1267.43233E12$, $12.78e-2$, 12 , -0 , 0 und INF ebenfalls Zahlenwerte dar.

Beispiel 5.1 (Konkatenation von (Zahlen-)Werten eines Dokuments)

Betrachtet man das nachfolgende Dokument

```
<person>
  <dob>1998</dob>
  <country>E</country>
  <age>20</age>
</person>
```

und den aus diesem Dokument entstandenen nachfolgenden Dokumentenbaum,



so ergibt sich, für den Fall, dass ein Pfad, z.B. //person, den person-Elementknoten lokalisiert, der (Zahlen-)Wert „1998E20“ aus der Konkatenation der Inhalte der Textknoten n_3 , n_5 und n_7 .

Die Aggregatfunktionen „*min*“ und „*max*“ sind von (reinen) Zahlenwerten als Inhalte der Textknoten eines Dokumentenbaums nicht zwingend betroffen, sondern erlauben es, aufgrund „fehlender Berechnung“ ihres Ergebnisses, auch auf *alphanumerische Zeichenketten*, d.h. auf die Inhalte von Textknoten eines Dokumentenbaums direkt, angewendet werden zu können.³ Ebenfalls ist die Aggregatfunktion „*count*“ von (reinen) Zahlenwerten oder alphanumerischen Zeichenketten für die Berechnung ihres Ergebnisses nicht betroffen. Sie besitzt nämlich als Ergebnis die *Anzahl Knoten*, d.h. die *Mächtigkeit* der *Knotenmenge*, die durch einen Pfad in einem Dokumentenbaum lokalisiert wird. Die Aggregatfunktion „*list*“ ist *keine Aggregatfunktion* des *XQuery-Standards* und besitzt als Ergebnis eine *Komma-separierte Liste* der einzelnen Inhalte aller Textknoten, die durch einen Pfad in einem Dokumentenbaum lokalisiert werden. Auch hier wird wieder, bevor ein Komma gesetzt wird, die Konkatenation der einzelnen Inhalte der Textknoten eines Dokumenten-

³Nach dem XPath- und XQuery-Standard in [38] ist es für die Aggregatfunktionen „*min*“ und „*max*“, um auf alphanumerischen Zeichenketten angewendet werden zu können, notwendig, dies explizit durch einen „(type)-cast“ anzugeben. So muss z.B. „fn:max(//person)“, falls alphanumerische Zeichenketten als Inhalte der durch den Pfad lokalisierten Textknoten zu erwarten sind, dies durch „fn:max(xs:string(//person))“ angegeben werden. Innerhalb der GCX Engine existiert kein „Typsystem“, weshalb das Vorliegen eines (reinen) Zahlenwerts oder einer alphanumerischen Zeichenkette für diese beiden Aggregatfunktionen dynamisch zur Laufzeit entschieden wird.

baums für den Fall vorgenommen, dass der durch einen Pfad lokalisierte Knoten einen Elementknoten dieses Dokumentenbaums darstellt.

Die im Sprachfragment XQ neben der Aggregatfunktion „*list*“ verfügbaren und ebenfalls nicht zum XQuery-Standard gehörenden Aggregatfunktionen „*stddev_samp*“ für die Standardabweichung der (Stich-)Probe (sample standard deviation), „*stddev_pop*“ für die Standardabweichung der Grundgesamtheit/Population (population standard deviation), „*var_samp*“ für die Varianz der (Stich-)Probe (sample variance) und „*var_pop*“ für die Varianz der Grundgesamtheit/Population (population variance) sind allesamt *statistische Aggregatfunktionen*. Für deren Berechnungen und Ergebnisse sei an dieser Stelle auf [24–28] verwiesen.

Zusammenfassend können also die Aggregatfunktionen „*sum*“, „*avg*“, „*stddev_samp*“, „*stddev_pop*“, „*var_samp*“ und „*var_pop*“ nur auf (reine) Zahlenwerte vom Datentyp „*xs:double*“ in [35], die Aggregatfunktionen „*min*“, „*max*“ und „*list*“ sowohl auf (reine) Zahlenwerte vom Datentyp „*xs:double*“ in [35] als auch auf alphanumerische Zeichenketten angewandt werden. Die Aggregatfunktion „*count*“ ist unabhängig von (reinen) Zahlenwerten oder alphanumerischen Zeichenketten, da für sie zum einen kein Ergebnis „berechnet“ wird und zum anderen diese nicht die Inhalte von Textknoten eines Dokumentenbaums benötigt.

Beim Versuch, eine Aggregatfunktion auf einen nicht für sie bestimmten „Typ“, d.h. (reiner) Zahlenwert und/oder alphanumerische Zeichenkette, anzuwenden, wird aufgrund der strikt sequenziellen Auswertung einer Anfrage, diese an der momentan bearbeiteten Stelle abgebrochen und ein Fehler, der auf eine fehlerhafte Anwendung bzw. Konvertierung hinweist, ausgegeben.

Des Weiteren ist im Rahmen der Erweiterung des Sprachfragments XQ um Aggregatfunktionen die Funktion „*exists*“ um die *Fähigkeit* ergänzt worden, auch eine durch eine for-Klausel zuvor definierte *Variable* als *Parameter* zu besitzen. Das Ergebnis bzw. der Wahrheitswert der Funktion „*exists*“ mit einer Variablen als Parameter, z.B. „*exists(\$x)*“, ist für alle Variablen stets wahr, da eine Variable zuvor durch eine for-Klausel an einen Knoten gebunden wurde und dieser aufgrund dessen auch stets vorhanden sein muss. In diesem Zuge sind ebenfalls die *for-Klauseln* erweitert worden, indem diese nun auch über den durch eine vorherige for-Klausel an eine Variable gebundenen *Knoten iterieren* können. Die Erweiterung der Iteration bzw. Bindung von Variablen in for-Klauseln über die Bindung eines Knotens einer

(anderen) Variablen einer vorherigen for-Klausel, z.B. „for \$y in \$x return α “, ist für das gesamte Sprachfragment XQ aus Gründen der Konsistenz für die Aggregatfunktionen, die als Parameter ebenfalls eine Variable besitzen können, z.B. „avg(\$x)“, erfolgt. Sie orientiert sich am XQuery-Standard, indem solche Konstrukte ebenfalls erlaubt sind. Dieser Zusammenhang von for-Klauseln und Aggregatfunktionen wird in der Idee zur Einbettung der Aggregatfunktionen im nächsten Abschnitt näher erläutert.

5.2 Idee zur Einbettung der Aggregatfunktionen

Der Textknoten⁴ eines Dokumentenbaums, dessen Inhalt bzw. (Zahlen-)Wert in die Berechnung des Ergebnisses einer Aggregatfunktion bereits eingeflossen ist, hat gegebenenfalls – sofort nachdem sein Inhalt erfasst worden ist – seine Relevanz für eine Anfrage verloren. Daher ist es sinnvoll bzw. zur Erfüllung der Forderung, den zur Auswertung einer Anfrage notwendigen Speicherbedarf minimal zu halten, notwendig, diesen Textknoten sofort (wieder) aus dem Speicher zu entfernen.

Die Idee, die auch Grundlage zur Einbettung der Aggregatfunktionen in das vorhandene Framework ist, basiert im Wesentlichen auf der Idee der in Abschnitt 3.4 beschriebenen Optimierung zum „früherem Entfernen von gespeicherten Knoten“. Diese schreibt, wie nachfolgendes Beispiel 5.2 verdeutlicht, alle Pfade in Ausgaben in *FR-Ausdrücke* um.

⁴Im Weiteren wird angenommen, dass der Pfad, der als Parameter einer Aggregatfunktion übergeben wird, ausschließlich Textknoten eines Dokumentenbaums lokalisiert. Dies stellt keine Einschränkung für die Idee und Einbettung der Aggregatfunktionen dar und kann daher auch auf den Fall, dass der Pfad einen Elementknoten eines Dokumentenbaums lokalisiert, übertragen werden.

Beispiel 5.2 (Anfrage für Quelltext E.1)

Die nachfolgende Anfrage gibt die Namen aller Entleiher aus.

```
<r> {
  for $person in /borrower/person return
    <name> {$person/name/text()} </name>
} </r>
```

Aus dieser Anfrage ergibt sich durch Anwendung der Optimierung zum „früherem Entfernen von gespeicherten Knoten“ die nachfolgende Anfrage.

```
<r> {
  for $person in /borrower/person return
    <name> {for $x in $person/name/text() return $x} </name>
} </r>
```

Durch dieses Umschreiben aller Pfade in Ausgaben in FR-Ausdrücke werden u.U. – abhängig von der dem ursprünglichen Pfad vorangehenden Variablen und deren erste direkte Vorgängervariablen (f_{sa}) –, wie nachfolgendes Beispiel 5.3 zeigt, *SignOff-Anweisungen* in diese eingefügt.

Beispiel 5.3 (Anfrage aus Beispiel 5.2 mit SignOff-Anweisungen)

Die Anfrage aus Beispiel 5.2 ergibt die nachfolgend umgeschriebene Anfrage mit SignOff-Anweisungen.

```
<r> {
  for $person in /borrower/person return
    (
      <name> {
        for $x in $person/name/text() return
          (
            $x,
            signOff($x, r2),
            signOff($x/dos::node(), r3)
          )
      } </name>,
      signOff($person, r1)
    )
} </r>,
signOff($root, r0)
```

Durch die Ausführung der *SignOff-Anweisungen* werden gegebenenfalls alle Rollen, die ein auszugebender Knoten mit seiner Speicherung erhalten hat, direkt nach seiner Ausgabe wieder entfernt. Falls dies erfolgt und der Knoten dadurch seine Relevanz für die weitere Anfrage verloren hat, d.h. er Definition 3.14 erfüllt, wird er, durch das Ausführen der von den SignOff-Anweisungen ebenfalls angestoßenen Speicherbereinigung, (wieder) aus dem Speicher entfernt.

Auf genau dieser Idee, wie nachfolgendes Beispiel 5.4 zeigt, beruht auch die *Einbettung* der *Aggregatfunktionen*. So werden die Variable oder der Pfad, die/der als Parameter einer Aggregatfunktion übergeben wurde, in einen *FR-Ausdruck* umgeschrieben.

Beispiel 5.4 (*Anfrage für Quelltext E.1*)

Die nachfolgende Anfrage gibt die (Gesamt-)Anzahl aller Entleiher, das Durchschnittsalter dieser, der von diesen insgesamt verfügbare Geldbetrag, den jüngsten Entleiher und das Datum des längst gültigen Accounts aus.⁵

```
<r> {
  <count> {count(//person)} </count>,
  <avg> {avg(//person/age)} </avg>,
  <sum> {sum(//person/account/amount)} </sum>,
  for $person in //person
    where ($person/age = min(//person/age)) return
      <name> {$person/name/text()} </name>,
  <max> {max(//person/account/valid/to)} </max>
} </r>
```

Für diese Anfrage ergibt sich die folgende umgeschriebene Anfrage mit FR-Ausdrücken der Pfade der Aggregatfunktionen, wobei die where-Klausel, wie in Abschnitt 3.1 beschrieben, direkt durch die Konstruktion eines bedingten Ausdrucks ersetzt worden ist.

```
<r> {
  <count> {count(for $a in //person return $a)} </count>,
  <avg> {avg(for $b in //person/age return $b)} </avg>,
  <sum> {sum(for $c in //person/account/amount return $c)} </sum>,
  for $person in //person return
    (
      if ($person/age = min(for $d in //person/age return $d)) then
        (
          <name> {$person/name/text()} </name>
        )
      else (),
    ),
  <max> {max(for $e in //person/account/valid/to return $e)} </max>
} </r>
```

Durch dieses Umschreiben werden nun, wie bei der Optimierung zum „früherem Entfernen von gespeicherten Knoten“, gegebenenfalls – abhängig von der dem ursprünglichen Pfad vorangehenden Variablen und deren erste direkte Vorgängerva-

⁵In dieser Anfrage ist zu beachten, dass die Inhalte der Textknoten dieses Dokumentenbaums für die Aggregatfunktion „max“, die das Datum des längst gültigen Accounts ausgibt, die Form „YYYY-MM-DD“ besitzen und diese Inhalte keine (reinen) Zahlenwerte darstellen.

riable (fsa) – SignOff-Anweisungen in die jeweiligen FR-Ausdrücke der einzelnen Aggregatfunktionen eingefügt. Dies führt dazu, dass ein bereits in die Berechnung des Ergebnisses einer Aggregatfunktion eingeflossener (Zahlen-)Wert bzw. der Textknoten, aus dessen Inhalt er stammt, nachdem er alle Rollen verloren hat, sofort durch die Ausführung der von den SignOff-Anweisungen angestoßenen Speicherbereinigung (wieder) aus dem Speicher entfernt wird.

Zur Berechnung des Ergebnisses einer Aggregatfunktion wird nun die im Körper der return-Klausel des FR-Ausdrucks stehende Variable der Ausgabe verwendet, um den Inhalt des Textknotens in jeder Iteration der for-Klausel zu ermitteln. Dazu wird in jeder Iteration der for-Klausel der an diese Variable gebundene Textknoten verwendet, um dessen Inhalt bzw. (Zahlen-)Wert an die for-Klausel zu „überreichen“. Die for-Klausel wiederum „übergibt“ diesen (Zahlen-)Wert an die jeweilige Aggregatfunktion, die diesen in das Ergebnis einrechnet. Nachdem dies erfolgt ist, wird die in der return-Klausel des jeweiligen FR-Ausdrucks stehende Sequenz von SignOff-Anweisungen ausgeführt und durch sie die entsprechende(n) Rolle(n) entfernt sowie die Speicherbereinigung aufgerufen. U.U. wird dadurch der Textknoten, dessen Inhalt bzw. (Zahlen-)Wert in der aktuellen Iteration der for-Klausel in das Ergebnis der Aggregatfunktion eingerechnet wurde, (wieder) aus dem Speicher entfernt. Dieses Vorgehen wird in jedem Iterationsschritt der for-Klausel wiederholt, so dass im Idealfall ein bereits in das Ergebnis einer Aggregatfunktion eingerechneter (Zahlen-)Wert bzw. der Textknoten, aus dessen Inhalt er stammt, sofort (wieder) aus dem Speicher entfernt wird.

Dieses Umschreiben des Parameters einer Aggregatfunktion in einen FR-Ausdruck hat, da der Parameter auch eine Variable sein kann, die zuvor im Sprachfragment XQ beschriebene Erweiterung von for-Klauseln notwendig gemacht.

5.3 Theoretische Grundlagen und Hintergründe

Die Einbettung der Aggregatfunktionen in die aktive Speicherbereinigung hat es erforderlich gemacht, die in Definition 4.2 genannten *Abhängigkeiten neu* zu fassen. Dies ist durch die nachfolgende Definition 5.1 erfolgt.

Definition 5.1 (Abhängigkeiten $dep_Q(\$x)$)

Sei Q eine Anfrage in XQ und $\$x \in Vars_Q$. Die Menge der *Abhängigkeiten der Variablen* $\$x$, bezeichnet mit $dep_Q(\$x)$, sind wie folgt definiert. Sei $\beta \preceq Q$ mit Rollenfunktion $r_Q(\beta) = r$, $for_{\$x}^{count}$ die Variable der Ausgabe des FR-Ausdrucks innerhalb der Aggregatfunktion „count“ und π_i ein Präfixpfad, dann

- $\langle \pi_{n-1}/axis::\nu[1], r \rangle \in dep_Q(\$x)$, falls $\beta = \text{„exists}(\$x/\pi_{n-1}/axis::\nu\text{“}$,
- $\langle \pi_{n-1}/axis::\nu/dos::node(), r \rangle \in dep_Q(\$x)$, falls β entweder ein Ausdruck in einer Ausgabe der Form „ $\$x/\pi_{n-1}/axis::\nu$ “, ein bedingter Ausdruck der Form „ $\$x/\pi_{n-1}/axis::\nu \text{ RelOp } \chi$ “ oder „ $\chi \text{ RelOp } \$x/\pi_{n-1}/axis::\nu$ “, und
- $\langle dos::node(), r \rangle \in dep_Q(\$x)$, falls β ein Ausdruck in einer Ausgabe der Form „ $\$x$ “ $\neq for_{\$x}^{count}$.

Da die in Definition 5.1 genannten Abhängigkeiten die Abhängigkeiten aus Definition 4.2 enthalten, werden diese in der weiteren Arbeit als $dep_Q(\$x)$ bezeichnet.

Beim *Vergleich* der beiden *Abhängigkeiten* in Definition 4.2 und in Definition 5.1 fällt auf, dass bis auf die Aggregatfunktion „count“ alle Aggregatfunktionen aufgrund ihrer Einbettung mit den bisherigen Abhängigkeiten aus Definition 4.2 harmonisiert hätten. Die Aggregatfunktion „count“ erforderte eine Redefinition, da sie bzw. der in ihr später umgeschriebene FR-Ausdruck durch die im Körper der return-Klausel stehende Variable der Ausgabe ein Tupel, in der durch die for-Klausel definierten Variablen zugehörigen Abhängigkeit, geschaffen hätte. Dies hätte dazu geführt, dass wesentlich mehr Knoten und Teilbäume eines Dokumentenbaums gespeichert worden wären als zur Berechnung des Ergebnisses der Aggregatfunktion „count“ notwendig gewesen wäre. Aus diesem Grund wird die Aggregatfunktion „count“ bzw. der in ihr nach dem Umschreiben des Parameters stehende FR-Ausdruck ohne die return-Klausel betrachtet, d.h. diese wird bei der Bestimmung der Abhängigkeiten einer Anfrage „ignoriert“. Dies wiederum führt dazu, dass im Projektionsbaum nur ein Knoten für die in der Aggregatfunktion „count“ stehende for-Klausel erzeugt wird. Somit werden nur die durch diesen Knoten des Projektionsbaums beschriebenen Pfad lokalisierten Knoten eines Dokumentenbaums gespeichert. Diese gespeicherten Knoten eines Dokumentenbaums sind für die Aus-

wertung der Aggregatfunktion „*count*“ vollkommen ausreichend.

Das Erzeugen eines Tupels in einer Abhängigkeit, was aufgrund des FR-Ausdrucks in einer Aggregatfunktion – mit Ausnahme der Aggregatfunktion „*count*“ – bzw. durch die im Körper der return-Klausel stehenden Variablen der Ausgabe dieses Ausdrucks erfolgt, führt in manchen Fällen zur Speicherung zu vieler Knoten eines Dokumentenbaums und wäre zur Berechnung des Ergebnisses einer Aggregatfunktion nicht notwendig. Dies liegt an dem durch die Variable der Ausgabe erzeugten Tupel der Form „/dos::node()“. So wird, für den Fall, dass die als Parameter einer Aggregatfunktion übergebene Variable an einen Elementknoten gebunden ist oder der Pfad einen Elementknoten lokalisiert, der komplette Teilbaum mit diesem Elementknoten als Wurzel gespeichert, obwohl nur die Textknoten von diesem für die Berechnung des Ergebnisses einer Aggregatfunktion notwendig wären. Die Umsetzung der Aggregatfunktion in dieser Form ist daher nur aus zwei Gesichtspunkten zulässig. Zum einen lokalisiert der Knotentest „node()“ eines Tupels der Form „/dos::node()“ auch Textknoten eines Dokumentenbaums, weshalb die zur Berechnung des Ergebnisses einer Aggregatfunktion benötigten (Inhalte der) Textknoten gespeichert werden und somit für die Berechnung des Ergebnisses vorliegen. Zum anderen erhält durch die Optimierung zum „Zuweisen von aggregierten Rollen“ nur der lokalisierte Elementknoten eine Rolle und „vererbt“ diese an alle seine Nachkommen, die somit durch die entsprechende SignOff-Anweisung auch nur von diesem wieder zu entfernen ist. Ein Abweichung, d.h. eine etwaige Deaktivierung oder das Entfernen dieser Optimierung, hätte zur Folge, dass zur Berechnung des Ergebnisses einer Aggregatfunktion zu viele Knoten gespeichert werden würden und dies der Minimierung des Speicherbedarfs der GCX Engine widerspräche.

Da innerhalb der GCX Engine kein „Typsystem“ existiert und alle intern verwendeten Werte als alphanumerische Zeichenketten behandelt werden, erfolgt die Bestimmung des Vorliegens eines (reinen) Zahlenwerts oder einer alphanumerischen Zeichenkette für die beiden Aggregatfunktionen „*min*“ und „*max*“ *dynamisch zur Laufzeit*. Dabei bestimmt der erste (Zahlen-)Wert, der einer dieser beiden Aggregatfunktionen von der for-Klausel „übergeben“ wird, beim Versuch der Überführung dieses in einen (reinen) Zahlenwert, den „Typ“, d.h. (reiner) Zahlenwert oder alphanumerische Zeichenkette, der restlichen Werte. Dieser wird somit vom ersten (Zahlen-)Wert einmalig festgelegt und für den Fall, dass dieser eine alphanumeri-

sche Zeichenkette ist, für alle weiteren nachfolgenden Inhalte der Textknoten unterstellt. Die Annahme eines (reinen) Zahlenwerts für alle weiteren nachfolgenden Inhalte der Textknoten wird im Falle, dass der erste Wert ein (reiner) Zahlenwert ist, getroffen. Ein „Wechsel“ der Inhalte von Textknoten von (reinen) Zahlenwerten auf alphanumerische Zeichenketten und umgekehrt wird während der Bestimmung des Minimums bzw. Maximums aufgrund der Tatsache, dass nach [38] alle in die Berechnung des Ergebnisses einer Aggregatfunktion einfließenden (Zahlen-)Werte einen gemeinsamen (Daten-)Typ besitzen müssen, ausgeschlossen. Folglich können zwar im Falle, dass der erste Wert eine alphanumerische Zeichenkette ist, weitere (reine) Zahlenwerte, die als alphanumerische Zeichenketten interpretiert werden, folgen, allerdings nicht umgekehrt, ohne einen (Konvertierungs-)Fehler hervorzurufen. Die Aggregatfunktion „*list*“ ist von dieser Bedingung, dass alle Werte nur einen gemeinsamen „Typ“ besitzen müssen, ausgenommen, da sie eine Komma-separierte Liste der Inhalte der Textknoten – ohne eine vorherige Prüfung dieser – als Ergebnis besitzt.

Aufgrund der Tatsache, dass die in diesem Kapitel beschriebene Idee und deren theoretische Ableitung in wesentlichen Zügen mit den vorgenommenen Implementierungen übereinstimmt, wird an dieser Stelle auf eine Beschreibung der praktischen Umsetzung der Aggregatfunktionen in der GCX Engine verzichtet.

Kapitel 6

GCX Optimierungen

Neben den bereits in Abschnitt 3.4 beschriebenen *vier Optimierungen* für die praktische Umsetzung der *aktiven Speicherbereinigung* in der *GCX Engine* sind im Rahmen dieser Arbeit *vier* (neue) *Optimierungen* hinzugefügt und die bereits bestehende Optimierung zum „Entfernen von redundanten Rollen eines Projektionsbaums“ erweitert worden. Wie alle bereits bestehenden Optimierungen haben auch die nun folgenden fünf Optimierungen das Ziel, den *Speicherbedarf* zu *reduzieren* und/oder die *Auswertungszeit* einer Anfrage zu *beschleunigen*. Alle fünf Optimierungen finden noch während der *statischen Analyse* statt und sind dabei in der zeitlichen Abfolge ihrer Anwendung erläutert. *Zwei Optimierungen* werden auf die in Definition 5.1 genannten *Abhängigkeiten*, *zwei* auf einen *Projektionsbaum* und *eine* als eine Art „Unterstützung“ der Speicherbereinigung, die durch die in einer Anfrage stehenden SignOff-Anweisungen angestoßen wird, angewandt.

6.1 Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden

Diese Optimierung wird auf die in Definition 5.1 genannten Abhängigkeiten bzw. die in ihnen enthaltenen Tupel angewandt. Grundsätzlich bestimmen Abhängigkeiten, welche Knoten eines Dokumentenbaums zur (korrekten) Auswertung einer Anfrage bzw. eines Ausdrucks, neben den ohnehin zu speichernden Knoten für die Iteration der for-Klauseln, zwingend benötigt werden. Dabei ist nicht auszuschließen, dass, wie das nachfolgende Beispiel 6.1 verdeutlicht, in *Abhängigkeiten für eine Variable* *Tupel* existieren, die *identische Pfade* besitzen.

Beispiel 6.1 (Anfrage für Quelltext F.1 mit zugehörigen Abhängigkeiten)

Die nachfolgende Anfrage gibt alle (geliehenen) Buchtitel und deren Entleiher aus, falls das jeweilige Buch einen Titel besitzt. Anschließend folgt die Ausgabe einer Liste aller E-Mail Adressen der Entleiher, falls der jeweilige Entleiher eine solche besitzt.

```
<r> {
  for $book in /loan/books/book
    where ($book/title/text() != "") return
      (
        <title> {$book/title/text()} </title>,
        <borrower> {$book//person/name} </borrower>
      ),
  for $email in //person/emailaddress
    where ($email != "") return
      $email
} </r>
```

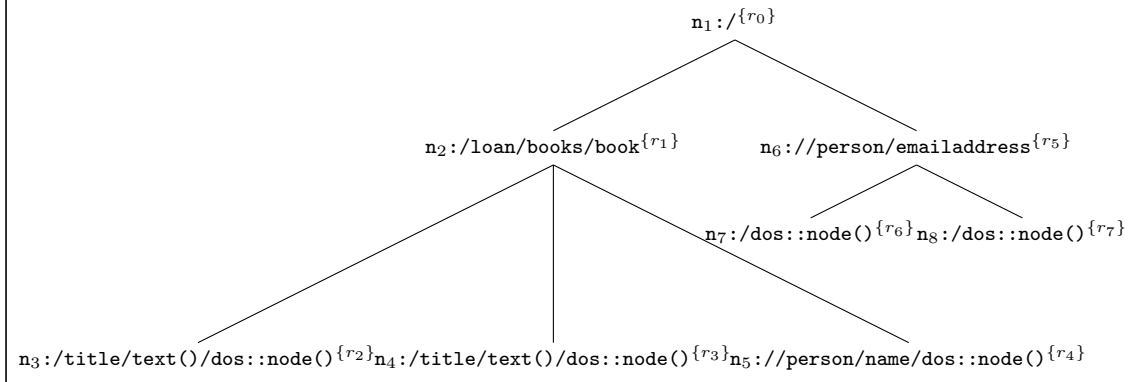
Dabei werden für die Variablen $\$book$ und $\$email$ die nachfolgenden Abhängigkeiten ermittelt.

$$\begin{aligned}
 dep_Q(\$book) &= \{ \langle child::title/child::text()/dos::node(), r_2 \rangle, \\
 &\quad \langle child::title/child::text()/dos::node(), r_3 \rangle, \\
 &\quad \langle descendant::person/child::name/dos::node(), r_4 \rangle \} \\
 dep_Q(\$email) &= \{ \langle dos::node(), r_6 \rangle, \langle dos::node(), r_7 \rangle \}
 \end{aligned}$$

Bei genauerer Betrachtung dieser beiden Abhängigkeiten bzw. ihrer Tupel fällt auf, dass das Tupel $\langle child::title/child::text()/dos::node() \rangle$ für die Variable $\$book$ und das Tupel $\langle dos::node() \rangle$ für die Variable $\$email$ doppelt vorkommen und lediglich das entsprechende Duplikat eine andere Rolle besitzt. Bei der Erstellung des Projektionsbaums werden, wie nachfolgendes Beispiel 6.2 dies verdeutlicht, die beiden jeweiligen Tupel bzw. ihre Pfade und Rollen als Kinder der Knoten des Variablenbaums, zu deren Variable sie gehören, eingefügt.

Beispiel 6.2 (Projektionsbaum mit Rollen für Anfrage aus Beispiel 6.1)

Die in Beispiel 6.1 enthaltene Anfrage ergibt den nachfolgend dargestellten Projektionsbaum mit Rollen.



Das hat zur Folge, dass die Knoten eines Dokumentenbaums, die sich in der Knotenmenge, die durch den beschriebenen Pfad des einen Knoten des Projektionsbaums, z.B. n_3 oder n_7 , lokalisiert wird, auch – aufgrund der Identität der (beschriebenen) Pfade – in der Knotenmenge, die durch den beschriebenen Pfad des anderen Knoten des Projektionsbaums, z.B. n_4 bzw. n_8 , lokalisiert wird, enthalten sind. Dadurch erhalten diese zu speichernden Knoten eines Dokumentenbaums auch (korrekterweise) beide Rollen, z.B. r_2 und r_3 oder r_6 und r_7 , zugewiesen. Die entsprechenden SignOff-Anweisungen zum Entfernen der jeweiligen Rollen stehen innerhalb der umgeschriebenen Anfrage mit SignOff-Anweisungen in einer Sequenz und werden somit aufgrund der strikt sequenziellen Auswertung einer Anfrage auch nacheinander ausgeführt. Das ist damit zu erklären, dass für alle Tupel von Abhängigkeiten einer Variablen stets dieselbe erste direkte Vorgängervariable (fsa) gilt. Das Zuweisen und das spätere Entfernen von (unterschiedlichen) Rollen, die sich aufgrund von Tupeln mit identischen Pfaden in Abhängigkeiten für eine Variable ergeben, ist überflüssig, weil dies lediglich zu mehr zu vergebenen Rollen der zu speichernden Knoten, zu mehr auszuführenden SignOff-Anweisungen zum Entfernen der entsprechenden Rollen und zu mehr Ausführungen der Speicherbereinigung führt.

Aus diesem Grund werden alle Tupel mit identischen Pfaden in Abhängigkeiten für eine Variable mit Hilfe dieser Optimierung eliminiert, so dass lediglich ein Tupel für den entsprechenden zuvor mehrmalig aufgetretenen identischen Pfad in

Abhängigkeiten für eine Variable erhalten bleibt.¹ Die Speicherung von Knoten eines Dokumentenbaums wird durch das Entfernen dieser Tupel nicht beeinflusst, weil stets eines der Tupel in Abhängigkeiten für eine Variable auch bei mehrmaligem Auftreten identischer Pfade erhalten bleibt.

Die praktische Umsetzung dieser Optimierung in der GCX Engine ist nach Erhalt aller Abhängigkeiten für alle Variablen für eine Anfrage durch das Untersuchen der Pfade aller Tupel aller Abhängigkeiten einer jeden Variablen auf identische Pfade erfolgt. Für den Fall, dass Tupel in Abhängigkeiten einer Variablen mit identischen Pfaden existieren, sind diese nach der Untersuchung entfernt worden. Der Vergleich der Identität zweier Pfade P und P' ist dabei auf die syntaktische Gleichheit der Pfade und somit auf die syntaktische Gleichheit von Achse und Knotentest, genauer ausgedrückt auf die Gleichheit der Achse und des Knotentests der einzelnen Schritte von P und P' an gleicher Position, zurückgeführt worden. Im Rahmen der Umsetzung ist diese Optimierung an ein globales „Deaktivierungs“-flag² gebunden worden, so dass sie, falls eine Optimierung nicht gewünscht wird, ausgeschaltet werden kann.

6.2 Entfernen von Tupeln in Abhängigkeiten bei enthaltenen Knotenmengen

Die auf die in Definition 5.1 genannten Abhängigkeiten angewandte zweite Optimierung entfernt die Tupel aus Abhängigkeiten, deren in ihnen stehenden Pfade enthaltene Knotenmengen lokalisieren. Das bedeutet, dass durch diese Optimierung alle Tupel aus Abhängigkeiten einer Variablen entfernt werden, deren Pfade Knotenmengen lokalisieren, die in der Knotenmenge, die durch den Pfad eines anderen Tupels dieser Abhängigkeiten lokalisiert wird, enthalten ist. Zunächst sei für diese Optimierung durch Definition 6.1 das Enthaltensein zweier oder mehrerer Pfade

¹Man beachte, dass dabei niemals Pfade von Tupeln aus verschiedenen Abhängigkeiten, d.h. aus Abhängigkeiten, die für verschiedene Variablen gelten, verglichen werden. Es werden stets nur die Pfade der Tupel auf Identität untereinander verglichen, die sich in gleichen Abhängigkeiten, d.h. in Abhängigkeiten einer Variablen, befinden.

²Das „Deaktivierungs“-flag zum Ausschalten der nicht mit dem Kern der Anwendung verbundenen Optimierungen ist dabei in Form eines Compiler-flags erfolgt.

bzw. die durch sie lokalisierte Knotenmengen definiert.

Definition 6.1 (Enthaltene Knotenmengen $P \supseteq P'$ von zwei Pfaden)

Seien P und P' zwei nicht zwingend verschiedene Pfade und T ein (beliebiger) Dokumentenbaum. Sei weiter $N_P(T)$ bzw. $N_{P'}(T)$ die durch den Pfad P bzw. P' lokalisierten Knotenmengen von T . Der Pfad P' ist in Pfad P *enthalten* bzw. die durch die beiden Pfade *lokalisierten Knotenmengen* sind *ineinander enthalten*, bezeichnet mit $P \supseteq P'$, falls $\forall(T) : N_P(T) \supseteq N_{P'}(T)$.

Falls $\forall(T) : N_P(T) \supseteq N_{P'}(T) \wedge N_P(T) \subseteq N_{P'}(T)$, bezeichnet mit $P \equiv P'$, so sind die beiden Pfade *äquivalent*.

Aufgrund der Tatsache, dass zu diesem Zeitpunkt, d.h. zum *Zeitpunkt* der *statischen Analyse*, das als *Datenstrom* erhaltene *Dokument* noch *nicht verarbeitet* worden ist, ist eine Bestimmung angesichts der in Definition 6.1 gemachten Aussage, ob ein Pfad in einem anderen Pfad enthalten ist, nicht *trivial*.

Für diese Optimierung wird zunächst einmal die im nachfolgenden Beispiel 6.3 enthaltene Anfrage und der ihr zugehörigen Abhängigkeiten betrachtet.

Beispiel 6.3 (Anfrage für Quelltext F.1 mit zugehörigen Abhängigkeiten)

Die nachfolgende Anfrage gibt alle namentlich genannten Entleiher und deren zu zahlende Gebühr aus.

```
<r> {
  for $borrower in //book/borrower
    where ($borrower/person/name != "") return
      (
        $borrower/person,
        $borrower/fee
      )
} </r>
```

Dabei werden für die Variablen $\$borrower$ die nachfolgenden Abhängigkeiten ermittelt.

$$\begin{aligned} dep_Q(\$borrower) = & \{ \langle child::person / child::name / dos::node(), r_2 \rangle, \\ & \langle child::person / dos::node(), r_3 \rangle \} \\ & \langle child::fee / dos::node(), r_4 \rangle, \end{aligned}$$

Bei genauerer Betrachtung der Abhängigkeiten der Variablen $\$borrower$ bzw. den in ihr enthaltenen Tupel fällt auf, dass für einen der Anfrage zugehörigen Doku-

mentenbaum, die durch den Pfad des Tupels $\langle child::person/dos::node() \rangle$ lokalisierte Knotenmenge größer ist als die durch den Pfad des Tupels $\langle child::person/child::name/dos::node() \rangle$. Daraus folgt, dass das Tupel mit der kleineren lokalisierten Knotenmenge überflüssig ist und entfernt werden kann. Der Grund, warum dies nur durch Betrachtung der Pfade der einzelnen Tupel erfolgen kann, liegt, wie auch für die Optimierung zum „Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden“, an der Tatsache, dass für die Pfade von Tupeln in einer Abhängigkeit die gleiche Variable ihren jeweiligen Pfaden vorangestellt ist.

Problematisch bei der Bestimmung des Enthaltenseins zweier Pfade ist die Forderung in Definition 6.1, dass das Enthaltensein zweier Pfade für alle Dokumente bzw. deren zugehörigen Dokumentenbäume gelten muss.

Für das Problem des Enthaltenseins zweier Pfade – ohne zugrunde liegendem Dokument bzw. Dokumentenbaum – existieren bereits Lösungsansätze. Daher ist für diese Optimierung auf einen der in [14] bzw. [15] beschriebenen Ansätze zurückgegriffen worden. Dort werden zwei Ansätze zur Bestimmung des Enthaltenseins zweier Pfade beschrieben. Der für diese Optimierung eingesetzte Ansatz aus [14] bzw. [15] ist auf Basis der benötigten Laufzeit entschieden worden. Die Wahl fiel auf den von der Laufzeit schnelleren Ansatz, dessen darin verwendeter *Algorithmus* für zwei Pfade P und P' mit $|P|$ bzw. $|P'|$ Schritten $\mathcal{O}(|P||P'|)$ benötigt, wohingegen der im zweiten Ansatz verwendete Algorithmus eine Laufzeit von $\mathcal{O}(|P|^2|P'|)$ hat. Neben der schnelleren Laufzeit des für diese Optimierung eingesetzten Ansatzes und der damit verbundenen besseren Effizienz, ist dieser, wie auch der zweite Ansatz, *korrekt (sound)*, d.h. falls „true“ das Ergebnis der Prüfung $P \supseteq P'$ für zwei Pfade P und P' , so gilt dies auch. Allerdings ist dieser, gegenüber dem zweiten Ansatz, *nicht vollständig (incomplete)*, d.h. in einigen Ausnahmefällen, für die an dieser Stelle auf [14] bzw. [15] verwiesen wird, wird durch ihn das Enthaltensein zweier Pfade nicht erkannt.

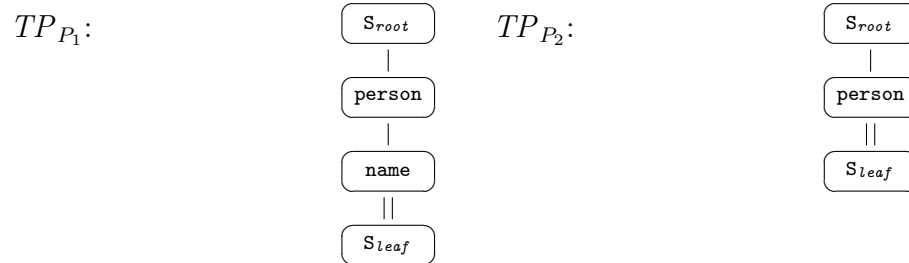
Die Gründe der Anwendung des schnelleren, aber nicht vollständigen Ansatzes, liegen zum einen an vielen durchzuführenden Vergleichen, worunter die Auswertungszeit einer Anfrage durch die GCX Engine zu leiden hätte, zum anderen an der Tatsache, dass es für diese Optimierung zu Gunsten der Laufzeit nicht erforderlich ist, alle enthaltenen Pfade zu erkennen.

Aufgrund der Komplexität des für diese Optimierung eingesetzten Ansatzes aus [14] bzw. [15] werden an dieser Stelle nur die wesentlichen Kernpunkte genannt, im Übrigen wird für eine vollständige Übersicht der Idee und für Beispiele auf [14] bzw. [15] verwiesen.

Das in [14] bzw. [15] betrachtete Sprachfragment, XP genannt, unterstützt bzw. betrachtet für das Enthaltensein zweier Pfade die Achsen „child“ und „descendant“, als Knotentests „*“ und einen Elementnamen sowie Prädikate. Zur Bestimmung des Enthaltenseins zweier Pfade wird der jeweilige Pfad in einen so genannten *Pfadbaum* (*tree pattern*) überführt. Ein solcher Pfadbaum, wie Beispiel 6.4 dies anschaulich zeigt, besitzt als Knoten die Knotentests der einzelnen Schritte eines Pfades und als Kanten die im jeweiligen Schritt des Knotentests verwendete Achse. Eine einfache Kante innerhalb eines Pfadbaums bezeichnet dabei die Achse „child“ und eine doppelte Kante die Achse „descendant“.

Beispiel 6.4 (*Erweiterter Pfadbaum*)

Für die zwei Pfade $P_1 = /child::person/child::name/dos::node()$ und $P_2 = /child::person/dos::node()$ ergeben sich die zwei nachfolgend dargestellten (erweiterten) Pfadbäume TP_{P_1} und TP_{P_2} .



Beispiel 6.4 zeigt im Vergleich zu den Pfadbäumen in [14] bzw. [15] schon deutlich die für diese Optimierung notwendigen *Veränderungen* und *Modifikationen*. So wird auch in [14] bzw. [15] vor der Überführung eines Pfades in einen Pfadbaum, um das Enthaltensein für einige („spezielle“) Pfade zu gewährleisten, ein *Schattenblatt*³ (*shadow leaf*), S_{leaf} , mit Achse „descendant“ als letzter Schritt eines Pfades bzw. in seinem zugehörigen Pfadbaum eingefügt.⁴ Da diese Optimierung nur auf

³Das Schattenblatt selbst entspricht dabei einem (beliebigen) Elementnamen, der für die Bestimmung des Enthaltenseins zweier Pfade in deren zugehörigen Pfadbäumen identisch sein muss.

⁴Für die nach dem Einfügen des Schattenblatts in einen Pfad vorgenommenen Umschreibungen zum Erhalt der so genannten „Adornments“ wird auf [14] bzw. [15] verwiesen.

die Pfade der Tupel in Abhängigkeiten angewandt wird, wird für Pfade, die als letzten Schritt „/dos::node()“ besitzen, dieser (letzte) Schritt dazu „umgewandelt“. Dadurch wird einerseits die nicht im Sprachfragment XP auftretende Achse „dos“ eliminiert, die grundsätzlich nur im letzten Schritt eines Pfades von Tupeln in Abhängigkeiten vorkommen kann. Andererseits wird der Schritt „/dos::node()“ beim Extrahieren der Tupel der Abhängigkeiten einer Anfrage an die (Original-)Pfade eingefügt. Der dadurch erhaltene (veränderte) Pfad eines Tupels in Abhängigkeiten entspricht somit nicht dem (Original-)Pfad der in einer Anfrage aufgetreten ist.

Neben diesem Schattenblatt, wie Beispiel 6.4 zeigt, wird als (weitere) Modifikation, die zur Umsetzung des Ansatzes notwendig war, eine *Schattenwurzel*⁵ (*shadow root*), S_{root} , einem Pfad bzw. seinem zugehörigen Pfadbaum „vorangestellt“, die dadurch den Wurzelknoten eines Pfadbaums bildet. Die Notwendigkeit dieser Modifikation liegt an den in [14] bzw. [15] betrachteten Pfaden, die stets *relativ* sind, wohingegen die in der GCX Engine verwendeten Pfade *absolut* sind.

Weiter existieren die für „Verzweigungen“ innerhalb eines Pfadbaums sorgenden Prädikate im Sprachfragment XQ nicht, weshalb die Pfadbäume im Gegensatz zu denen in [14] bzw. [15] stark vereinfacht sind. Dafür existieren im Sprachfragment XQ die Knotentests „node()“ und „text()“, die im Sprachfragment XP nicht vorkommen. Der Knotentest „node()“ ist daher für alle Schritte mit Ausnahme des letzten Schrittes durch den Knotentest „*“ ersetzt worden, da der Knotentest „node()“ für diese Schritte dem Knotentest „*“ entspricht. Pfade, die den Knotentest „text()“ in einem Schritt mit Ausnahme des letzten Schrittes besitzen, lokalisieren keine Knoten eines Dokumentenbaums, weshalb dieser Fall für das Enthaltensein zweier Pfade auf eine Prüfung, die ebenfalls nicht Bestandteil des Ansatzes aus [14] bzw. [15] ist, reduziert werden konnte. Der letzte Schritt eines Pfades ist für die vom Sprachfragment XP nicht abgedeckten Knotentests, „node()“ und „text()“, anhand von Prüfungen, die ebenfalls nicht Bestandteil des Ansatzes aus [14] bzw. [15] sind, auf das Enthaltensein geprüft worden. Diese sind dazu zuvor aus beiden Pfaden entfernt worden und nach der Bestimmung auf das Enthaltensein dieser beiden „verkürzten“ Pfade mit einer Prüfung des Enthaltenseins des zuvor entfernten

⁵Wie bereits für das Schattenblatt entspricht die Schattenwurzel ebenfalls einem (beliebigen) Elementnamen, der für die Bestimmung des Enthaltenseins zweier Pfade in deren zugehörigen Pfadbäumen identisch sein muss.

letzten Schrittes entschieden worden.

Nach dem Erhalt zweier Pfadbäume für zwei Pfade wird nun ein *Homomorphismus* von dem einen in den anderen Pfadbaum mit dem in [14] bzw. [15] genannten *Algorithmus* gesucht. Falls ein solcher existiert, so sind die beiden Pfade ineinander enthalten.

Beim Vergleich dieser Optimierung und der Optimierung zum „Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden“ wird man feststellen, dass identische Pfade von dieser Optimierung eingeschlossen werden, d.h. das Enthaltensein gilt natürlich auch für zwei identische Pfade. Die Aufteilung in zwei Optimierungen erfolgte aufgrund der für die jeweilige Optimierung benötigten Laufzeit. So benötigt die Bestimmung des Enthaltenseins zweier Pfade P und P' mit $|P|$ bzw. $|P'|$ Schritten $\mathcal{O}(|P||P'|)$. Hingegen benötigt die Optimierung zum „Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden“ lediglich $\mathcal{O}(|P|)$ und bildet zur Einsparung von Laufzeit – da sie vor dieser hier beschriebenen Optimierung angewandt wird – eine Art „*Vorselektion*“ für die Bestimmung des Enthaltenseins zweier Pfade durch diese Optimierung.

Diese Optimierung ist ebenfalls an das globale „Deaktivierungs“-flag gebunden worden und kann daher, falls eine Optimierung nicht gewünscht wird, ausgeschaltet werden. Aufgrund der Komplexität und der sehr vereinfachten Darstellung dieses formalen Ansatzes in diesem Abschnitt wird an dieser Stelle auf eine Beschreibung der vorgenommenen Implementierungen in der GCX Engine verzichtet.

6.3 Entfernen von nicht benötigten Knoten eines Projektionsbaums

Die dritte Optimierung wird direkt auf einen Projektionsbaum angewandt und behandelt die Knoten eines Projektionsbaums, deren Pfade einen Schritt mit beliebiger Achse und dem Knotentest „`text()`“ besitzen. Dabei wird die Tatsache, dass *Textknoten* eines jeden *Dokumentenbaums* stets *Blätter* darstellen, ausgenutzt, um nicht benötigte Knoten eines Projektionsbaums zu identifizieren und diese dann zu

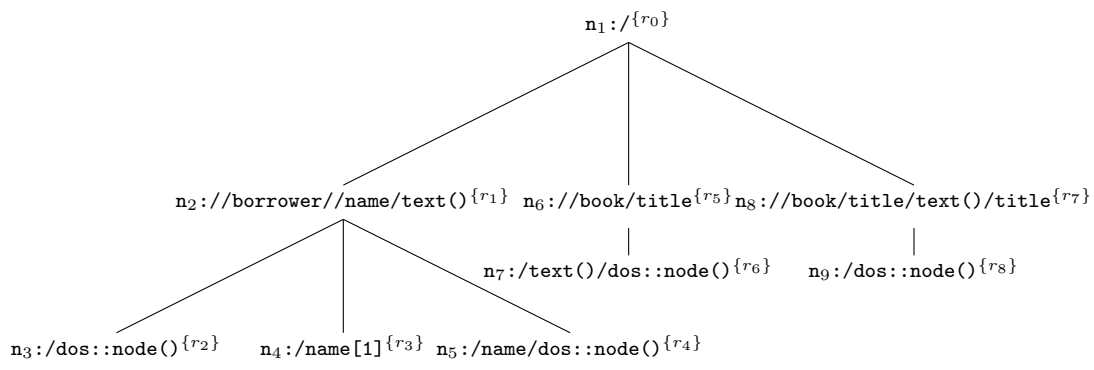
entfernen.⁶ Das nachfolgende Beispiel 6.5 verdeutlicht dies anhand einer Anfrage mit Pfaden, die einen Schritt mit Knotentest „text()“ besitzen.

Beispiel 6.5 (Anfrage für Quelltext F.1 mit zugehörigem Projektionsbaum)

Die nachfolgende Anfrage besitzt Pfade, die keine Knoten eines (beliebigen) Dokumentenbaums lokalisieren, und gibt daher, obwohl mehr Ausdrücke und Ausgaben in ihr enthalten sind, für alle (geliehenen) Bücher lediglich die Namen ihrer Entleiher, gefolgt von den Titeln aller (geliehenen) Bücher aus.

```
<r> {
  for $borrower in //borrower//name/text() return
    (
      <name> {$borrower} </name>,
      if (exists($borrower/name)) then <na> {$borrower/name} </na> else ()
    ),
  for $books in //book/title return
    <title> {$books/text()} </title>,
  for $title in //book/title/text()/title return
    <na> {$title} </na>
} </r>
```

Aus dieser Anfrage ergibt sich der nachfolgend dargestellte Projektionsbaum mit Rollen.



Dabei lokalisieren die durch Knoten n_4 , n_5 , n_8 und n_9 beschriebenen Pfade des Projektionsbaums keine Knoten eines dieser Anfrage zugehörigen Dokumentenbaums, da *Textknoten* eines solchen *keine Nachkommen* besitzen. Der durch Knoten n_3 beschriebene Pfad des Projektionsbaums lokalisiert aufgrund der Achse „dos“ mit Knotentest „node()“ zwar Knoten eines dieser Anfrage zugehörigen Dokumenten-

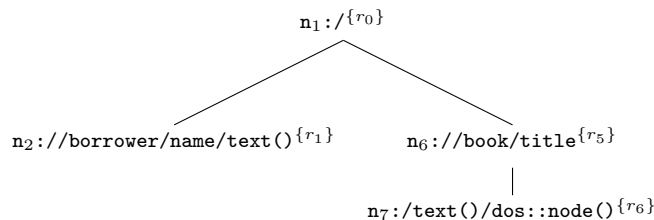
⁶Man beachte, dass zu diesem Zeitpunkt bzw. zum Zeitpunkt der statischen Analyse weder das als Datenstrom erhaltene Dokument gelesen wurde noch begonnen wurde, die Anfrage auszuwerten. Aus diesem Grund ist es nicht möglich, weitere Knoten eines Projektionsbaums zu entfernen, falls der durch einen Knoten des Projektionsbaums beschriebene Pfad ebenfalls keine Knoten (eines einer Anfrage zugehörigen) Dokumentenbaums lokalisiert.

baums, ist aber in Bezug auf die Funktion eines Projektionsbaums, der die für eine Anfrage notwendig zu speichernden Knoten eines Dokumentenbaums bestimmt, überflüssig. Das ist damit zu erklären, dass Knoten n_3 bereits durch den in Knoten n_2 beschriebenen Pfad des Projektionsbaums abgedeckt wird und lediglich zu einer weiteren zu vergebenden Rolle den zu speichernden Knoten, die sich in der Knotenmenge des durch Knoten n_2 beschriebenen Pfades befinden, führt.

Daher werden nach der Erstellung eines Projektionsbaums durch diese Optimierung, wie der im nachfolgenden Beispiel 6.6 dargestellte (optimierte) Projektionsbaum zeigt, Knoten eines Projektionsbaums, deren Pfade einen Schritt mit Knotentest „text()“ enthalten, diesen aber nicht als Knotentest des letzten Schrittes besitzen, samt all ihrer Nachkommen entfernt. Hingegen verbleiben Knoten eines Projektionsbaums in diesem, deren letzter Schritt ihres Pfades den Knotentest „text()“ besitzen, und verlieren nur all ihre Nachkommen. Eine Ausnahme hiervon bilden allerdings – und nur für diesen Fall – die aus den Tupeln aus Abhängigkeiten gewonnenen Knoten eines Projektionsbaums, die vor ihrem letzten Schritt „/dos::node()“ ihres Pfades einen Schritt mit Knotentest „text()“ besitzen dürfen.

Beispiel 6.6 (*Optimierter Projektionsbaum mit Rollen aus Beispiel 6.5*)

Nach Anwendung dieser Optimierung ergibt sich für den Projektionsbaum aus Beispiel 6.5 der nachfolgend dargestellte Projektionsbaum mit Rollen.



Die praktische Umsetzung dieser Optimierung in der GCX Engine ist durch einen *Tiefendurchlauf* des Projektionsbaums erfolgt. Bei diesem werden die einzelnen Schritte der Pfade eines jeden Knotens des Projektionsbaums auf den Knotentest „text()“ untersucht. Falls ein Knoten einen Schritt mit Knotentest „text()“ in seinem Pfad besitzt, dieser Schritt allerdings nicht der letzte dieses Pfades darstellt, wird dieser Knoten und sein kompletter Teilbaum, von dem er die Wurzel ist, aus dem Projektionsbaum entfernt. Analog mit dem Unterschied gilt, dass ein Knoten im Projektionsbaum verbleibt und nur sein kompletter Teilbaum, dessen Wurzel er

ist, entfernt wird, falls dieser Knoten einen Schritt mit Knotentest „text()“ am Ende seines Pfades besitzt. Wie auch die zuvor genannte Optimierung ist auch die hier beschriebene Optimierung an das globale „Deaktivierungs“-flag gebunden worden und kann daher ebenfalls, falls eine Optimierung nicht gewünscht wird, ausgeschaltet werden.

Die Grundidee dieser Optimierung, dass *Textknoten* eines jeden *Dokumentenbaums* stets *Blätter* darstellen, hat auch zwischen den beiden *Komponenten* – *Anfrageevaluator* und *Speichermanager* – bei der Anforderung neuer Knoten durch eine *getNext(\$x/π)*-Anfrage (Request) ihre Anwendung gefunden. Die Suche im Speicher nach lokalisierten Knoten für einen Pfad wird dabei sofort beendet bzw. gar nicht erst vorgenommen, falls (1) ein Pfad einen Schritt mit Knotentest „text()“ besitzt, dieser aber nicht der letzte Schritt dieses Pfades darstellt, oder (2) die einem Pfad vorangehende Variable im Moment an einen Textknoten gebunden ist. Der Grund für die zweite Bedingung, deren Tatsache nicht alleine nur wegen dem Knotentest „text()“, sondern auch im Falle des Knotentests „node()“ des letzten Schrittes eines Pfades auftreten kann, liegt an den zwei vom Sprachfragment XQ unterstützten vorwärtsgerichteten Achsen „descendant“ und „child“, die aus Sicht der aktuellen Bindung einer Variablen an einen Textknoten niemals einen weiteren Knoten lokalisieren können. Zwar existiert die Achse „descendant“, die auch aus Sicht der aktuellen Bindung einer Variablen an einen Textknoten diesen lokalisieren kann, jedoch tritt diese – falls überhaupt – nur in Pfaden von Tupeln in Abhängigkeiten auf und ist keine vom Sprachfragment XQ direkt unterstützte bzw. verwendbare Achse in einer Anfrage.

6.4 Entfernen von redundanten Rollen eines Projektionsbaums

Die bereits in Abschnitt 3.4 beschriebene Optimierung zum „Entfernen von redundanten Rollen eines Projektionsbaums“ hat in manchen Fällen die Rollen von Knoten eines Projektionsbaums als redundant erkannt und diese entfernt. Das *Erkennen* von *redundanten Rollen* eines Projektionsbaums und das damit verbundene *Entfernen* von diesen ist allerdings nur für manche Anfragen und nur in wenigen

Ausnahmefällen erfolgt. Redundante Rollen eines Projektionsbaums existieren allerdings für eine Vielzahl von Anfragen bzw. lassen sich in den Knoten der aus diesen Anfragen entstandenen Projektionsbäumen entfernen. Aus diesem Grund ist diese Optimierung erweitert worden. Zunächst wird dazu in Definition 6.2 festgehalten, wann die Rolle eines Knotens eines Projektionsbaums redundant ist.

Definition 6.2 (*Redundante Rolle(n) eines Projektionsbaums*)

Sei Q eine Anfrage in XQ, vt der aus Q entstandene Variablenbaum und vt_n ein Knoten dieses Variablenbaums. Sei weiter pt der aus vt entstandene Projektionsbaum und pt_n ein Knoten dieses Projektionsbaums. Sei weiter P_n der Pfad von pt_n und r_n die Rolle von pt_n . Die Rolle r_n eines Knotens des Projektionsbaums pt_n – und im Falle der Erfüllung von Bedingung (2) ebenfalls die Rollen seines kompletten Teilbaums, dessen Wurzel er ist, – ist/sind *redundant*, falls eine oder auch beide der folgenden Bedingungen erfüllt ist/sind, wobei beide Bedingungen für einen Knoten des Projektionsbaums pt_n stets zu prüfen sind und Bedingung (2) stärker als Bedingung (1) ist, d.h. den Vorzug auch im Falle einer erfüllten Bedingung (1) erhält:

- (1) pt_n ist ein innerer Knoten des Projektionsbaums.
- (2) pt_n stammt von vt_n und besitzt einen Geschwisterknoten $pt_{n'}$, dessen letzter Schritt seines Pfades $P_{n'}$ „/dos::node()“ ist und es gilt weiter: $P_{n'} \supseteq P_n$ ⁷

Aufgrund der stärkeren Bedingung (2) innerhalb dieser Definition und der Tatsache, dass diese die Rolle eines Knotens des Projektionsbaums pt_n und die Rollen seines kompletten Teilbaum als redundant einstuft, würde der Projektionsbaum einen Teilbaum mit Wurzel pt_n ohne Rollen und somit ohne Funktion besitzen. Aus diesem Grund wird – falls Bedingung (2) erfüllt ist – dieser mit redundanten Rollen versehene Teilbaum inklusive dem Knoten pt_n direkt aus dem Projektionsbaum wieder entfernt.

⁷Wie bereits bei der Optimierung zum „Entfernen von Tupeln in Abhängigkeiten bei enthaltenen Knotenmengen“ gilt es zu beachten, dass der letzte Schritt „/dos::node()“ eines Pfades als Schattenblatt (shadow leaf) seines entsprechenden Pfadbaums „umgewandelt“ wird und nicht zur Bestimmung des Enthaltenseins zweier Pfade miteinbezogen wird.

An dieser Stelle muss gesagt werden, dass beide in Definition 6.2 genannten Bedingungen auf einem in der Implementierung der GCX Engine wichtigen Mechanismus, dem so genannten „*Sperrmechanismus*“, beruhen. Dieser „Sperrmechanismus“ war bereits in der Implementierung des Ausgangspunktes dieser Arbeit vorhanden. Er ist damals eingefügt worden, um zu verhindern, dass die im Moment an Variablen gebundenen Knoten nicht aus dem Speicher entfernt werden. Dies tritt grundsätzlich immer auf, da eine oder auch mehrere SignOff-Anweisung dafür Sorgen können, dass alle Nachkommen von einem an eine Variable gebundenen Knoten keine Rollen mehr besitzen und somit durch Ausführung der Speicherbereinigung entfernt werden. Dies würde allerdings dazu führen, dass der aktuell an eine Variable gebundene Knoten – falls er selbst ebenfalls keine Rollen mehr besitzt – entfernt werden würde bzw. streng nach Definition 3.14 – die irrelevante Knoten des Dokumentenbaums bestimmt – entfernt werden müsste. Dies ist aber, um Komplikationen mit der Speicherbereinigung zu vermeiden, durch eine „Sperrung“ der aktuell an eine Variable gebundenen Knoten verhindert worden. Erst nachdem die Variable an einen neuen Knoten gebunden wird, wird die „Sperrung“ auf dem ursprünglich an diese Variablen gebundenen Knoten aufgehoben und auf ihm die Speicherbereinigung zum Entfernen dieses Knotens aus dem Speicher aufgerufen. Somit führt das „Sperrung“ und das damit verbundene Verhindern des Entfernens der aktuell an eine Variable gebundenen Knoten aus dem Speicher lediglich zu einer kurzen Verzögerung der Speicherbereinigung.

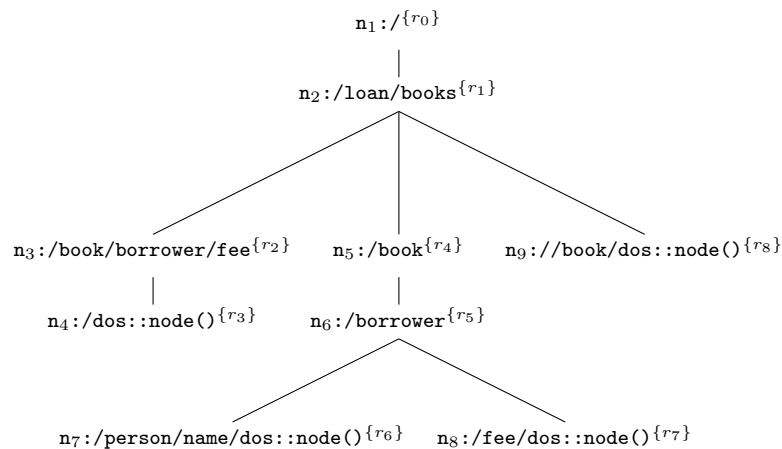
Zur Verdeutlichung der beiden in Definition 6.2 genannten Bedingungen, sei die im nachfolgenden Beispiel 6.7 enthaltene Anfrage und der ihr zugehörige Projektionsbaum betrachtet.

Beispiel 6.7 (Anfrage für Quelltext F.1 mit zugehörigem Projektionsbaum)

Die nachfolgende Anfrage gibt die Gesamtsumme aller zu zahlender Gebühren der Entleiher, gefolgt von deren Namen und deren jeweilig zu zahlender Gebühr aus. Anschließend werden die Informationen aller (geliehenen) Bücher vollständig ausgegeben. Hierbei ist der Pfad innerhalb der Aggregatfunktion „sum“ direkt durch eine FR-Ausdruck ersetzt worden.

```
<r> {
  for $books in /loan/books return
    (
      sum(for $x in $books/book/borrower/fee return $x),
      for $book in $books/book return
        for $borrower in $book/borrower return
          (
            $borrower/person/name,
            $borrower/fee
          ),
      $books//book
    )
} </r>
```

Aus dieser Anfrage ergibt sich der nachfolgend dargestellte Projektionsbaum mit Rollen.



Die erste Bedingung in Definition 6.2 lässt sich mit Definition 3.14, die irrelevante Knoten eines Dokumentenbaums identifiziert, und anhand des Algorithmus in Quelltext 3.2, der das Entfernen von Rollen und die Speicherbereinigung vollzieht, begründen. Ein Knoten ist demnach irrelevant und wird durch das Ausführen der Speicherbereinigung entfernt, falls er und auch kein Nachkommen von ihm eine

Rolle mehr inne haben. Demnach muss vor Ausführung der Speicherbereinigung⁸, d.h. vor Ausführung des Algorithmus in Quelltext 3.2, lediglich dafür Sorge getragen werden, dass es (relevante) Knoten im Dokumentenbaum mit einer Rolle gibt, die sich tiefer im Dokumentenbaum befinden, als Knoten, die noch relevant sind, allerdings aufgrund des Entfernens von redundanten Rollen der Knoten eines Projektionsbaums keine Rolle zugewiesen bekommen haben. Damit wird sichergestellt, dass noch relevante Knoten, die aufgrund des Entfernens von redundanten Rollen eines Projektionsbaums keine Rolle(n) erhalten haben, nicht durch die Ausführung der Speicherbereinigung entfernt werden.

Jeder Knoten eines Projektionsbaum beschreibt einen Pfad, der die benötigten bzw. zu speichernden Knoten eines Dokumentenbaums zur (korrekten) Auswertung einer Anfrage bzw. eines Ausdrucks festlegt. Die beschriebenen Pfade der Knoten eines Projektionsbaums ergeben sich dabei durch die *Konkatenation* der einzelnen Pfade der Knoten des Projektionsbaums von der Wurzel „/“ zu einem Knoten. Durch das Entfernen aller Rollen von allen inneren Knoten des Projektionsbaums dieser Optimierung behalten die Blätter eines Projektionsbaums stets ihre Rolle. Somit erhalten auch nur die durch die beschriebenen Pfade der Blätter eines Projektionsbaums lokalisierten Knoten eines Dokumentenbaums eine Rolle zugewiesen. Diese Knoten eines Dokumentenbaums sind zudem die tiefsten in diesem, die überhaupt eine Rolle erhalten hätten und stellen nun wiederum durch den Erhalt einer Rolle sicher, dass kein Knoten, der sich über diesen Knoten im Dokumentenbaum befindet, durch die Ausführung der Speicherbereinigung entfernt wird. Somit werden nach Anwendung der ersten Bedingung in Definition 6.2 die Rollen der Knoten n_1 , n_2 , n_3 , n_5 und n_6 des in Beispiel 6.7 enthaltenen Projektionsbaums wieder entfernt.

Die zweite Bedingung in Definition 6.2 beruht auf vielen Tatsachen und Grundlagen. Zunächst einmal besitzt jeder Knoten des Projektionsbaums, der ehemals ein Knoten des Variablenbaums war, eine erste direkte Vorgängervariable (fsa). Diese erste direkte Vorgängervariable (fsa) ist auch indirekt für die Knoten des Projek-

⁸Dabei gilt es zu beachten, dass die Speicherbereinigung nur lokal begrenzt in einem Dokumentenbaum ausgeführt wird und auch ein Dokumentenbaum bei Ausführung der Speicherbereinigung nicht global betrachtet wird. Die Speicherbereinigung wird dabei nur so lange ausgeführt, so lange irrelevante Knoten durch das bottom-up des Dokumentenbaums entfernt werden können.

tionsbaums gültig, die aus den Tupeln der Abhängigkeiten entstanden sind, da die in den Tupeln enthaltenen Pfade und Rollen als Kinder der ihren zugehörigen Variable, d.h. des ihrer Variable zugehörigen (ehemaligen) Knotens des Variablenbaums, angehängt werden. Damit besitzen zwei Geschwisterknoten eines Projektionsbaums auch dieselbe erste direkte Vorgängervariable (fsa), unabhängig davon, ob sie ein (ehemaliger) Knoten des Variablenbaums waren oder der Knoten sind, der aus einem Tupel in Abhängigkeiten stammt. Die erste direkte Vorgängervariable (fsa) wiederum bestimmt die Position bzw. zugehörige for-Klausel einer Anfrage, für die die einem Knoten des Projektionsbaums bzw. für dessen Pfad und Rolle eine SignOff-Anweisung eingefügt werden soll. Somit ist die Position bzw. zugehörige for-Klausel einer Anfrage die gleich, in denen SignOff-Anweisungen zum Entfernen der Rollen für zwei Geschwisterknoten eines Projektionsbaums eingefügt werden. Diese SignOff-Anweisungen stehen somit in einer Sequenz und werden nun aufgrund der strikt sequenziellen Auswertung einer Anfrage auch nacheinander ausgeführt.

Tatsache ist auch, dass die ersten direkten Vorgängervariablen (fsa) der Knoten eines Teilbaums des Projektionsbaums niemals eine Vorgängervariable der ersten direkten Vorgängervariablen (fsa) der Wurzel dieses Teilbaums darstellen. Das bedeutet, dass für die Anfrage Q in Beispiel 6.7 $fsa_Q(\$borrower) \not\prec_Q fsa_Q(\$book)$ gilt.⁹ Somit werden in der umgeschriebenen Anfrage mit SignOff-Anweisungen die SignOff-Anweisungen der Wurzel eines Teilbaums des Projektionsbaums aufgrund der strikt sequenziellen Auswertung einer Anfrage niemals vor SignOff-Anweisungen ausgeführt, die für einen Knoten dieses Teilbaums in eine Anfrage eingefügt wurden.

Weitere Tatsache für Bedingung zwei in Definition 6.2 ist, dass der für das Enthaltensein herangezogene Geschwisterknoten als letzten Schritt „/dos::node()“ besitzt. Damit werden aufgrund der Bestimmung des Enthaltenseins der Pfade zweier Geschwisterknoten alle weiteren Knoten des zu entfernenden Teilbaums von ihm inkludiert, d.h. die durch die einzelnen beschriebenen Pfade der Knoten dieses entfernten Teilbaums des Projektionsbaums können niemals mehr als den zur Bestimmung des Enthaltenseins herangezogenen (Geschwister-)Knoten lokalisieren. Grund hierfür sind die zwei vom Sprachfragment XQ unterstützten vorwärtsgerichteten Achsen, die niemals mehr als den verbleibenden Knoten mit letztem Schritt „/dos::node()“

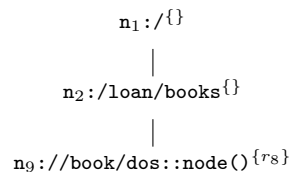
⁹Ursache hierfür ist der Zusammenhang von Elternvariablen (parVar) und ersten direkten Vorgängervariablen (fsa), auf den an dieser Stelle nicht weiter eingegangen wird.

lokalisieren können.

Knoten des Projektionsbaums in Beispiel 6.7, die zur Erfüllung der zweiten Bedingung in Definition 6.2 in Betracht kommen, sind n_3 , n_5 und n_9 . Sei P_{n_i} der Pfad des Knotens n_i . So gilt $P_{n_9} \supseteq P_{n_3}$ und $P_{n_9} \supseteq P_{n_5}$, d.h. die durch die Pfade P_{n_3} und P_{n_5} lokalisierten Knotenmengen sind in der durch Pfad P_{n_9} lokalisierten Knotenmenge enthalten. Daher werden, wie nachfolgendes Beispiel zeigt, diese zwei Teilbäume inklusive ihrer Wurzel (n_3 und n_5) aus dem Projektionsbaum entfernt und es ergibt sich der im nachfolgenden Beispiel 6.8 dargestellt (optimierte) Projektionsbaum.

Beispiel 6.8 (*Projektionsbaum mit redundanten Rollen für Beispiel 6.7*)

Der in Beispiel 6.7 dargestellte Projektionsbaum ergibt nach Anwendung der in Definition 6.2 genannten zwei Bedingungen den nachfolgend dargestellten Projektionsbaum mit der noch verbleibenden Rolle in seinem Blatt n_9 .



Durch Anwendung dieser Optimierung wird ein Projektionsbaum einerseits stark reduziert, andererseits wird ein Großteil der zuvor vergebenen Rollen der Knoten eines Projektionsbaums wieder entfernt. Das führt letztlich dazu, dass weniger Rollen den zu speichernden Knoten eines Dokumentenbaums vergeben und weniger SignOff-Anweisungen in eine Anfrage eingefügt werden und somit weniger Ausführungen der Speicherbereinigung erfolgen.

Die praktische Umsetzung dieser Optimierung in der GCX Engine ist durch einen *Tiefendurchlauf* des Projektionsbaums erfolgt. Dabei sind die beiden Bedingungen in Definition 6.2 neben dem Enthaltensein zweier Pfade mit Hilfe der existierenden Zeiger der Knoten von Eltern auf Kinder eines Projektionsbaums geprüft worden. Problematisch erwies sich dabei die erste Bedingung in Definition 6.2, die nicht, wie definiert, umgesetzt werden konnte. Bei ihr besteht das Problem der Ausführung der Speicherbereinigung, nachdem eine „Sperrung“ auf einem Knoten des Dokumentenbaums aufgehoben worden ist. Dabei konnte die Grundannahme der ersten Bedingung in Definition 6.2, dass jeder Teilbaum des Dokumentenbaums

auch einen Knoten, der in der Knotenmenge enthalten ist, die durch den in einem Blatt des Projektionsbaums beschriebene Pfad lokalisiert wird, für einige Anfrage nicht gehalten werden. Ursache hierfür ist der Fall, dass ein solcher Knoten eines Dokumentenbaums, der durch den Erhalt einer Rolle relevante Knoten „schützt“, die aufgrund des Entfernens von redundanten Rollen eines Projektionsbaums keine Rolle erhalten haben, nicht als Knoten eines Teilbaums des Dokumentenbaums existiert. Da alle Knoten, solange die Speicherbereinigung nicht ausgeführt wird, im Speicher verbleiben, werden sie durch die Ausführung der Speicherbereinigung nach Aufhebung der Sperre(n) entfernt, da sie selbst keine Rolle(n) erhalten haben und kein Knoten mit einer Rolle als Nachkommen existiert, der daran hindert, diese aus dem Speicher zu entfernen. Aus diesem Grund ist die erste Bedingung in Definition 6.2 abgeschwächt worden, so dass nur die Rollen aller inneren Knoten entfernt werden solange die für diese Knoten stehende for-Klausel in ihrer return-Klausel keine „direkte Ausgabe“ besitzt. Eine direkte Ausgabe ist dabei z.B. die Ausgabe von (alphanumerischen) Zeichenketten oder Elementen, die direkt in einer Sequenz von Ausdrücken der return-Klausel erfolgen.

Auch diese Optimierung ist an das globale „Deaktivierungs“-flag gebunden worden und kann daher, falls eine Optimierung nicht gewünscht wird, ausgeschaltet werden.

6.5 Fixierung der Auswertungsreihenfolge von SignOff-Anweisungen

Eine weitere Optimierung besteht in einer Art „Unterstützung“ der Speicherbereinigung, die durch die in einer Anfrage enthaltenen SignOff-Anweisungen angestoßen wird. Eine SignOff-Anweisung, „signOff($\$/\pi, r$)“, entfernt grundsätzlich die in ihr enthaltene Rolle r von den durch Pfad „ $\$/\pi$ “ lokalisierten Knoten eines Dokumentenbaums. Alle dabei lokalisierten Knoten werden direkt nach dem Entfernen einer Rolle darauf überprüft, ob sie gemäß Definition 3.14 irrelevant sind, um sie so früh als möglich wieder aus dem Speicher zu entfernen. Betrachtet man nochmals die in Beispiel 6.1 enthaltene Anfrage und deren im nachfolgenden Beispiel 6.9 dargestellte zugehörige umgeschriebene Anfrage mit SignOff-Anweisungen, so fällt

auf, dass in Sequenzen von SignOff-Anweisungen einer Variablen zunächst die Rolle des Knotens, an den eine Variable aktuell gebunden ist, z.B. „signOff(\$book, r₁)“, entfernt und für diesen Knoten die Speicherbereinigung aufgerufen wird. Erst nachdem dies erfolgt ist, werden die dieser SignOff-Anweisung weiter folgenden SignOff-Anweisungen ausgeführt.

Beispiel 6.9 (Anfrage aus Beispiel 6.1 mit SignOff-Anweisungen)

Die in Beispiel 6.1 enthaltene Anfrage ergibt die nachfolgend dargestellte umgeschriebene Anfrage mit SignOff-Anweisungen, wobei die where-Klauseln, wie in Abschnitt 3.1 beschrieben, direkt durch die Konstruktion von bedingten Ausdrücken ersetzt worden sind.

```
<r> {
  for $book in /loan/books/book return
    (
      if ($book/title/text() != "") then
        (
          <title> {$book/title/text()} </title>,
          <borrower> {$book//person/name} </borrower>
        )
      else (),
      signOff($book, r1),
      signOff($book/title/text()/dos::node(), r2),
      signOff($book/title/text()/dos::node(), r3),
      signOff($book//person/name/dos::node(), r4)
    ),
  for $email in //person/emailaddress return
    (
      if ($email != "") then
        (
          $email
        )
      else (),
      signOff($email, r5),
      signOff($email/dos::node(), r6),
      signOff($email/dos::node(), r7)
    )
} </r>,
signOff($root, r0)
```

Die Erfolgsaussicht, dass durch das Entfernen der Rolle des Knotens der aktuellen Bindung einer Variablen in einer Sequenz von SignOff-Anweisungen dieser Knoten nach Definition 3.14 irrelevant ist, ist eher gering. Ursache hierfür sind die für das Entfernen weiterer Rollen verantwortlichen nachfolgenden SignOff-Anweisungen, z.B. „signOff(\$book/title/text()/dos::node(), r₂)“. Diese entfernen Rollen an den ge-

speicherten Knoten eines Dokumentenbaums, die sich aufgrund der zwei vorwärtsgerichteten Achsen des Sprachfragments XQ, stets tiefer im Dokumentenbaum befinden als der Knoten an den eine Variable aktuell gebunden ist.

Die *Einfüge-* und somit *Auswertungsreihenfolge* der SignOff-Anweisungen in einer Anfrage ist bisher nach den zugeteilten Rollen der Knoten des Projektionsbaums erfolgt, d.h. nach der sich durch einen Tiefendurchlauf des Projektionsbaums ergebenden Zuweisung der Rollen für die einzelnen Knoten des Projektionsbaums. Durch eine *Fixierung* der Einfüge- und somit Auswertungsreihenfolge von SignOff-Anweisungen in einer Anfrage ist es jedoch möglich, die Erfolgsaussichten, mögliche irrelevante Knoten, die durch den in einer SignOff-Anweisung enthaltenen Pfad lokalisiert werden, zu erhöhen. Dazu wird versucht, zunächst die Rollen zu entfernen, die den Knoten vergeben wurden, die sich innerhalb eines Dokumentenbaums am tiefsten befinden. Hierdurch würde der Dokumentenbaum dann sukzessive bottom-up aus dem Speicher entfernt und das Überprüfen relevanter Knoten im Speicher, die noch Relevanz für die Auswertung einer Anfrage aufgrund von Nachkommen, die eine Rolle haben, verhindert werden. Der Gewinn hiervon wäre, dass die Suche nach Nachkommen, die eine Rolle besitzen, entfallen kann.

Da die Bestimmung der Tiefe eines durch einen Pfad lokalisierten Knotens während der statischen Analyse aufgrund fehlenden Wissens über den Aufbau und die Form des Dokumentenbaums nicht möglich ist, erfolgt die Fixierung der Einfüge- und somit Auswertungsreihenfolge der SignOff-Anweisungen anhand von „Distanzen“ der einzelnen in ihnen enthaltenen Pfaden. Die Basis dieser „Distanzen“ beruht dabei auf der *Gewichtung* der Achsen aller Schritte und dem Knotentest des letzten Schrittes eines Pfades. Dazu ist der Achse „child“ eine Gewichtung von *eins*, der Achse „descendant“ und „dos“ eine Gewichtung von *drei* zugeteilt worden. Die Gewichtung des Knotentests des letzten Schrittes eines Pfades ist für „*“ und einen Elementnamen auf *zwei*, für „node()“ auf *drei* und für „text()“ auf *vier* festgelegt worden. Die Gründe für die Festlegung genau dieser Gewichtungen liegen für die Achsen an der Tatsache, dass – unter der Annahme, dass die Achse „descendant“ bzw. „dos“ nicht als Ersatz für die Achse „child“ verwendet wird – die Achse „descendant“ und die Achse „dos“ Knoten lokalisieren, die sich tiefer im Dokumentenbaum befinden (können) als die durch die Achse „child“ lokalisierten Knoten. Die Gründe bezüglich der Gewichtungen des Knotentests des letzten Schrittes beruhen zum

einen auf der Tatsache, dass der letzte Schritt eines Pfades das Resultat des gesamten Pfades festlegt, zum anderen, dass die Knotentests „*“ und einem Elementnamen die Elementknoten eines Dokumentenbaums lokalisieren, die grundsätzlich niemals Blätter eines Dokumentenbaums sind. Der Knotentest „node()“ lokalisiert alle Knotentypen, zu denen auch Textknoten gehören können. Allerdings können sich aufgrund keiner Einschränkung an den Knotentyp darunter auch Elementknoten befinden, die keine Blätter eines Dokumentenbaums darstellen. Die Blätter – und nur diese – und somit die tiefsten Knoten eines Dokumentenbaums werden durch den Knotentest „text()“ lokalisiert, weshalb ihm die höchste Gewichtung zugeteilt wurde.

Das Gesamtgewicht eines Pfades P mit $n \geq 1$ Schritten wird aus den Einzelgewichten seiner Schritte p_i mit $1 \leq i \leq n$ letztlich mit Hilfe der nachfolgenden Gewichtungsfunktion $w(P)$ ermittelt, wobei $w_{axis}(p_i)$ die Gewichtung der Achse und $w_{nodetest}(p_i)$ die Gewichtung des Knotentests des Schrittes p_i bezeichnen.

$$w(P) = \left(\sum_{i=1}^{n-1} w_{axis}(p_i) \right) + w_{axis}(p_n) \cdot w_{nodetest}(p_n)$$

Der „leere“ Pfad in SignOff-Anweisungen, d.h. SignOff-Anweisungen, die nur eine Variable enthalten und somit nur einen Knoten lokalisieren, erhalten das Gesamtgewicht *null*.

Das durch die Gewichtungsfunktion $w(P)$ erhaltene Gesamtgewicht eines Pfades P ergibt sich aus der Summe der Gewichte der Achsen aller seiner Schritte mit Ausnahme des letzten Schrittes plus dem Gewicht der Achse multipliziert mit dem Gewicht des Knotentests des letzten Schrittes. Die geringere Gewichtung bzw. die nicht mit eingerechneten Knotentests aller vor dem letzten Schritt befindlichen Schritte eines Pfades wurde vorgenommen, da der Knotentest für diese Schritte eines Pfades irrelevant ist bzw. der Knotentest „node()“ für diese Schritte mit dem Knotentest „*“ gleichzusetzen ist und dieser wiederum einem beliebigen Elementnamen entspricht. Der Knotentest „text()“ spielt als Knotentest aller vor dem letzten Schritt eines Pfades befindlichen Schritte grundsätzlich keine Rolle, da ein solcher

Pfad keine Knoten eines (beliebigen) Dokumentenbaums lokalisiert.¹⁰

Mit Hilfe des Gesamtgewichts eines Pfades wird innerhalb der SignOff-Anweisungen einer Variablen die Einfüge- und somit die Auswertungsreihenfolge dieser in eine Anfrage bestimmt und es ergibt sich, wie nachfolgendes Beispiel 6.10 zeigt, eine fixierte Auswertungsreihenfolge der SignOff-Anweisungen einer Variablen.

¹⁰Man beachte, dass durch das Deaktivieren der (weiteren) Optimierungen, besonders der Optimierung zum „Entfernen von nicht benötigten Knoten eines Projektionsbaums“, Pfade, die den Knotentest „text()“ nicht als letzten Schritt besitzen, auftreten können.

Beispiel 6.10 (Anfrage aus Beispiel 6.9 mit SignOff-Anweisungen)

Mit Hilfe der Gewichtungsfunktion $w(P)$ ergeben sich für die Pfade in den SignOff-Anweisungen aus der in Beispiel 6.9 enthaltenen Anfrage für die Variablen $\$book$, die nachfolgenden Gesamtgewichte

$w(\$book) = 0$; $w(\$book/title/text()/dos :: node()) = 11$ und

$w(\$book//person/name/dos :: node()) = 13$.

Für die Pfade der SignOff-Anweisungen der Variablen $\$email$ ergeben sich die nachfolgenden Gesamtgewichte

$w(\$email) = 0$ und $w(\$email/dos :: node()) = 9$.

Somit ergibt sich als Resultat die nachfolgend dargestellte Anfrage mit fixierter Auswertungsreihenfolge der SignOff-Anweisungen.

```
<r> {
  for $book in /loan/books/book return
  (
    if ($book/title/text() != "") then
    (
      <title> {$book/title/text()} </title>,
      <borrower> {$book//person/name} </borrower>
    )
    else (),
    signOff($book//person/name/dos::node(), r4),
    signOff($book/title/text()/dos::node(), r2),
    signOff($book/title/text()/dos::node(), r3),
    signOff($book, r1)
  ),
  for $email in //person/emailaddress return
  (
    if ($email != "") then
    (
      $email
    )
    else (),
    signOff($email/dos::node(), r6),
    signOff($email/dos::node(), r7),
    signOff($email, r5)
  )
} </r>,
signOff($root, r0)
```

Die Fixierung der Einfüge- und somit Auswertungsreihenfolge von SignOff-Anweisungen beruht natürlich auf der Annahme, dass das Gesamtgewicht eines Pfades auch die Tiefe seiner lokalisierten Knoten eines Dokumentenbaums ausdrückt. Falls sich dennoch eine nicht der Tiefe der durch die Pfade lokalisierten Knoten eines Dokumentenbaums entsprechende Auswertungsreihenfolge von SignOff-Anweisungen

ergeben sollte, so ist dies zu diesem Zeitpunkt nicht besser entscheidbar und kann auch durch diese Optimierung nicht erreicht werden. Vielmehr soll sie der Speicherbereinigung eine „Unterstützung“ bieten, so dass mögliche irrelevante Knoten gemäß Definition 3.14 zuerst lokalisiert und diese somit vor allen weiteren noch zu diesem Zeitpunkt relevanten Knoten, früher aus dem Speicher entfernt werden.

Die praktische Umsetzung dieser Optimierung in der GCX Engine ist durch die Berechnung des Gesamtgewichts durch Anwendung der Gewichtungsfunktion $w(P)$ für jeden Pfad einer jeden SignOff-Anweisung erfolgt. Mit Hilfe des Gesamtgewichts eines Pfades wird unter Verwendung von *Bubblesort* die Einfüge- und somit Auswertungsreihenfolge der ihm zugehörigen SignOff-Anweisung, zu der auch ihre Rolle gehört, in einer Anfrage fixiert. Im Gegensatz zu den anderen in diesem Kapitel beschriebenen Optimierungen ist diese Optimierung nicht an das globale „Deaktivierungs“-flag gebunden, sondern mit dem Kern der Anwendung verschmolzen worden. Ursache hierfür ist die Optimierung zum „Entfernen von redundanten Rollen eines Projektionsbaums“, die im Falle ihrer Anwendung einen Großteil aller Rollen entfernt und die verbleibenden Rollen bzw. die ihnen zugehörigen SignOff-Anweisungen nicht (mehr) fixiert werden müssen bzw. eine Fixierung aufgrund der geringen Anzahl Rollen nicht mehr notwendig ist.

6.6 Gegensätzliche Optimierungen

Die meisten in dieser Arbeit (neu) hinzugefügten Optimierungen stehen in einem Widerspruch zur bereits existierenden Optimierung zum „früherem Entfernen von gespeicherten Knoten“. Diese Optimierung, mittels der bereits gespeicherte Knoten u.U. früher aus dem Speicher entfernt werden, verhält sich konträr zu den vier Optimierungen „Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden“, „Entfernen von Tupeln in Abhängigkeiten bei enthaltenen Knotenmengen“, „Fixierung der Auswertungsreihenfolge von SignOff-Anweisungen“ und zur zweiten Bedingung in Definition 6.2, die die redundanten Rollen eines Projektionsbaums bestimmt. Bei der Optimierung zum „früherem Entfernen von gespeicherten Knoten“ werden, wie nachfolgendes Beispiel 6.11 zeigt, alle Pfade in Ausgaben durch die Konstruktion von FR-Ausdrücken ersetzt. Dadurch werden gegebenenfalls –

abhängig der dem ursprünglichen Pfad vorangehenden Variablen und deren ersten direkten Vorgängervariablen (fsa) – SignOff-Anweisungen in diese eingefügt und bei der durch die SignOff-Anweisungen angestoßenen Speicherbereinigung Knoten früher (wieder) aus dem Speicher entfernt.

Beispiel 6.11 (*Anfrage aus Beispiel 6.1 mit früherem Entfernen*)

Die in Beispiel 6.1 enthaltene Anfrage ergibt nach Anwendung der Optimierung zum „früherem Entfernen von gespeicherten Knoten“, die im nachfolgend dargestellt Anfrage mit Optimierung, wobei auch hier die where-Klauseln direkt durch die Konstruktion von bedingten Ausdrücken ersetzt worden sind.

```
<r> {
  for $book in /loan/books/book return
    if ($book/title/text() != "") then
      (
        <title> {for $x in $book/title/text() return $x} </title>,
        <borrower> {for $y in $book//person/name return $y} </borrower>
      )
    else (),
  for $email in //person/emailaddress return
    if ($email != "") then
      $email
    else ()
} </r>
```

Dadurch ergeben sich für die zwei neuen Variablen $\$x$ und $\$y$ auch zwei neue Mengen von Abhängigkeiten, was das Entfernen von Tupeln in Abhängigkeiten durch die zwei Optimierungen „Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden“ und „Entfernen von Tupeln in Abhängigkeiten bei enthaltenen Knotenmengen“ nicht mehr möglich macht, da diese beiden Optimierungen nur auf Tupel in Abhängigkeiten einer Variablen angewendet werden bzw. generell nur auf diese angewendet werden dürfen. Die Optimierung „Fixierung der Auswertungsreihenfolge von SignOff-Anweisungen“ ist u.U. insofern davon betroffen, als durch das Umschreiben der Pfade in Ausgaben und das damit verbundene Einführen von neuen Variablen durch for-Klauseln ehemalige Sequenzen von SignOff-Anweisungen einer Variablen aufgelöst werden. Die Folge ist, dass keine oder nur eine bedingte Fixierung der Einfüge- und somit Auswertungsreihenfolge einer ursprünglichen Sequenz von SignOff-Anweisungen einer Variablen mehr möglich ist. Die redundanten Rollen von Teilbäumen und das Entfernen dieser samt deren Wurzel, die sich aufgrund der zweiten Bedingung in Definition 6.2 ergeben würde, sind insofern betroffen, als

in einem Projektionsbaum der Knoten, der den ursprünglich aus einem Tupel einer Abhängigkeit stammenden Pfad besitzt, nun durch einen Knoten mit dem Pfad der for-Klausel und einem Kind mit dem Pfad aus dem Tupel der Abhängigkeit für die Variable dieser for-Klausel „ersetzt“ wird. Somit wird der zuvor mögliche Vergleich zwischen Geschwisterknoten aufgelöst und dadurch eine Erfüllung von Bedingung zwei aus Definition 6.2 verhindert.

Kapitel 7

Experimentelle Resultate

Nachdem die Theorie und Praxis des Ausgangspunktes der aktiven Speicherbereinigung und deren Implementierung in der GCX Engine in Kapitel 3, die Erweiterungen dieser beiden um Pfade mit mehreren Schritten in Kapitel 4 und um Aggregatfunktionen in Kapitel 5 sowie die neu hinzugefügten Optimierungen in Kapitel 6 beschrieben wurden, befasst sich dieses Kapitel mit den im Rahmen dieser Arbeit durchgeführten *Experimenten*. In diesen ist die GCX Engine für ein breites Spektrum von Anfragen auf Dokumente mit unterschiedlichen Größen, die sich von 10 MB bis 200 MB erstrecken, getestet worden. Dabei ist sie mit anderen XQuery Engines und auch zu sich selbst, d.h. zur (ursprünglichen) Implementierung des Ausgangspunktes und zu der im Rahmen dieser Arbeit erfolgten (aktuellen) Implementierung mittels unterschiedlichen Konfigurationen, verglichen worden, um den *Vorteil* der *aktiven Speicherbereinigung*, d.h. den Vorteil der Kombination von *statischer* und *dynamischer Analyse*, zu untermauern. Um einen Vergleich zwischen den einzelnen XQuery Engines untereinander zu ermöglichen, ist hierzu auf die zur Auswertung einer Anfrage auf ein Dokument benötigte *Zeit* und der dazu notwendige *Speicherbedarf* zurückgegriffen worden. Hierfür ist das bekannte *XML Benchmark Project* (*XMark*) von [23] verwendet worden, das eine Benchmark-Umgebung für XQuery zur Verfügung stellt. Im nun Folgenden werden zunächst die für die Experimente notwendigen technischen und XML- bzw. XQuery-spezifischen Grundlagen der Versuchsreihen beschrieben und anschließend die erhaltenen Messwerte der Auswertungszeit und des Speicherbedarfs der einzelnen XQuery Engines präsentiert und diskutiert.

7.1 Grundlagen und Aufbau

Die für die Versuchsreihen verwendeten Dokumente sind mit Hilfe des *XMark-Daten/Dokumenten-Generators* von [23] erzeugt worden und werden im Weiteren als *XMark-Dokumente* bezeichnet. Diese XMark-Dokumente umfassen die Größen 10 MB, 50 MB, 100 MB und 200 MB. Aufgrund der Einschränkung der GCX Engine, keine Attribute in Dokumenten zu unterstützen, sind alle Attribute der erzeugten XMark-Dokumente entfernt und durch die Konstruktion neuer (Unter-)Elemente ersetzt worden. So ist ein (ursprüngliches) Element der Form `<tag attr = "val">... </tag>` zu `<tag><tag_attr>val</tag_attr>... </tag>` umgeschrieben worden. Weitere Veränderungen waren nicht notwendig und die nach dem Umschreiben aller Attribute gewonnenen XMark-Dokumente sind selbstverständlich für alle XQuery Engines verwendet worden.

Die auf die XMark-Dokumente verwendeten Anfragen, die für alle Versuchsreihen und – falls ein Vergleich vorgenommen wurde – für alle XQuery Engines Anwendung fanden, stammen ebenfalls von [23] und werden – wie bereits die XMark-Dokumente – im Weiteren als *XMark-Anfragen* bezeichnet. Alle XMark-Anfragen sind leicht modifiziert oder ergänzt worden, falls die Auswertung dieser aufgrund des einer XQuery Engine zugrunde liegenden Sprachfragments nicht möglich war. So sind, aufgrund des Umschreibens aller Attribute in den verwendeten XMark-Dokumenten, die Schritte eines Pfades mit Achse „attribute“ (kurz: „@“) und einem Attributnamen als Knotentest durch die Achse „child“ und dem für den Attributnamen verwendeten (Unter-)Elementnamen als Knotentest ersetzt worden. Des Weiteren wurden alle Prädikate, die in den Schritten eines Pfades der XMark-Anfragen auftraten, entfernt und alle let-Klauseln der XMark-Anfragen, falls dies möglich war, umgeschrieben. Die somit für die Versuchsreihen erhaltenen XMark-Anfragen sind in Unterabschnitt G.1.1 und Unterabschnitt G.1.2 aufgelistet. Dabei sind, um einen Vergleich zwischen der (ursprünglichen) Implementierung des Ausgangspunktes und der im Rahmen dieser Arbeit erfolgten (aktuellen) Implementierung zu ermöglichen, die in Unterabschnitt G.1.2 enthaltenen XMark-Anfragen mit Pfaden, die mehrere Schritte besitzen, und in Unterabschnitt G.1.1 die äquivalenten XMark-Anfragen mit Pfaden, die einen Schritt besitzen, zu finden. Die in Unterabschnitt G.1.1 enthaltenen XMark-Anfragen sind aus den in Unterabschnitt G.1.2

enthaltenen XMark-Anfragen durch Umschreiben aller Pfade mit mehreren Schritten in Pfade mit einem Schritt, sowohl für Pfade der for-Klauseln als auch für Pfade in Ausgaben, gewonnen worden. So ist, wie nachfolgendes Beispiel 7.1 dies exemplarisch für die XMark-Anfrage Q01 zeigt, für den Fall, dass eine XMark-Anfrage die gleiche Nummer besitzt und diese in beiden Abschnitten zu finden ist, sie das entsprechende Pendant.

Beispiel 7.1 (*XMark-Anfrage Q01*)

Die aus Unterabschnitt G.1.2 stammende nachfolgende XMark-Anfrage Q01

```
<query01> {
  for $person in /site/people/person
    where($person/person_id = "person0") return
      $person/name/text()
} </query01>
```

ergibt durch Umschreiben der Pfade mit mehreren Schritten der Pfade der for-Klauseln und der Pfade in Ausgaben in Pfade mit einem Schritt für diese beiden die aus Unterabschnitt G.1.1 stammende nachfolgende XMark-Anfrage Q01

```
<query01> {
  for $site in /site return
    for $people in $site/people return
      for $person in $people/person
        where($person/person_id = "person0") return
          for $name in $person/name return
            $name/text()
} </query01>
```

Dabei sind in Unterabschnitt G.1.1 die XMark-Anfragen, die nach dem Umschreiben ein anderes oder abweichendes Resultat gegenüber der entsprechenden XMark-Anfrage aus Unterabschnitt G.1.2 lieferten, weggelassen worden.

Als XQuery Engines kamen im Gegensatz zu den (ursprünglichen) Versuchsreihen in [20] bzw. [21] *Qizx/open Version 2.1* von [39] und *Saxon Version 9.1.0.1J* von [19] zum Zuge. Die Gründe, nur diese zwei XQuery Engines zum Vergleich heranzuziehen, haben unterschiedliche Ursachen. Zum einen besitzen diese, wie auch die GCX Engine, die Fähigkeit, Anfragen auf großen Dokumenten auszuwerten. Zum anderen unterstützt *FluXQuery* von [7] bzw. in [5, 6] Pfade mit Achse „descendant“ nicht. *Galax* von [17] als Referenz(-implementierung) des XQuery-Standards ist zwar eine in-memory Engine, lieferte allerdings bereits in den damaligen Versuchsreihen keine/-n kompetitiven Zeiten und Speicherbedarf, da das Hauptziel dieser XQuery Engine mehr auf der vollständigen Implementierung des XQuery-Standards als auf

einer Optimierung der Auswertungszeit und des Speicherbedarfs liegt. Dennoch ist Galax zum Vergleich der Resultate der XMark-Anfragen auf die XMark-Dokumente für die GCX Engine herangezogen worden, damit keine falschen oder abweichenden Resultate der XMark-Anfragen die Auswertungszeit und den Speicherbedarf verfälschen.¹ Die noch fehlende XQuery Engine *MonetDB/XQuery* von [8] schied als Kandidat wegen der Verwendung eines sekundären Speichers, d.h. die Verwendung einer physikalischen Datenbank und Auswertung einer Anfrage mittels relationaler Techniken, die mittels Indizes und Ähnlichem erfolgen, aus.

Neben der Sicherstellung der Korrektheit der Resultate der XMark-Anfragen auf die XMark-Dokumente ist für die GCX Engine eine eigens entworfene *Testanfrageumgebung* zum Einsatz gekommen. Diese hat sich im Gegensatz zu der für *GCX Engine Version 1.0b* verwendeten Testanfrageumgebung von damals ca. 200 Anfragen auf rund 900 Anfragen für *GCX Engine Version 2.0* erhöht. Die Testanfrageumgebung von *GCX Engine Version 2.0* enthält die unterschiedlichsten Anfragen für die vom Sprachfragment XQ unterstützten Konstrukte und Ausdrücke, die sich von der Konstruktion von Elementen, über bedingte Ausdrücke, Pfade mit einem Schritt und Pfade mit mehreren Schritten bis zu den in diesen Versuchsreihen verwendeten XMark-Anfragen erstrecken. Dabei ist nicht nur die Korrektheit der Resultate der einzelnen Anfragen dieser Testanfrageumgebung und der XMark-Anfragen auf die XMark-Dokumente sichergestellt worden, sondern auch die in Definition 3.8 geforderten Bedingungen, die die Sicherheit bei der Auswertung einer Anfrage mit SignOff-Anweisungen gewährleisten soll, geprüft worden. So waren für alle XMark-Anfragen auf allen XMark-Dokumenten stets alle zu entfernenden Rollen definiert und alle Knoten nach Beendigung der Auswertungen aller XMark-Anfragen aus dem Speicher entfernt worden, so dass dieser leer war.

¹Wegen der innerhalb der XQuery Engines vorgenommen Rundung(en) der Resultate von Aggregatfunktionen sind diese im Vergleich zur GCX Engine leicht abweichend, allerdings ist dies lediglich auf Rundungsfehler zurückzuführen. Die Korrektheit der Resultate der XMark-Anfragen ist dabei auf die für die Auswertung einer XMark-Anfrage auf ein XMark-Dokument verantwortlichen Bestandteile, z.B. Anzahl Iterationen der for-Klauseln, Ausgaben von Knoten und Teilbäumen bzw. der ihnen entsprechenden Elemente samt Inhalt etc., die sich wesentlich auf die Auswertungszeit einer XMark-Anfrage niederschlagen, sichergestellt.

Alle Versuchsreihen sind auf einem *Desktop PC* mit einer auf *2.13 GHz* getakteten *Intel Core2 Duo E6400 CPU*, dem *3 GB DDR2* mit *667 MHz nonECC RAM* Arbeitsspeicher und eine *250 GB Hitachi P7K500 SATA-II* Festplatte mit *8 MB Cache* zur Verfügung stehen, ausgeführt worden. Als Betriebssystem kam die Linux-Distribution *Ubuntu, Version 7.10 Gutsy Gibbon* zum Einsatz. Für die beiden zum Vergleich herangezogenen Java basierten XQuery Engines, *Qizx/open Version 2.1* und *Saxon Version 9.1.0.1J*, ist die *Java Runtime Environment (JRE) Version 1.6.0 Update 04* verwendet worden. Die maximale Größe des *Heaps* bei der Initialisierung der *Java Virtual Machine (JVM)* ist durch die Option „*-Xmx2600m*“ für diese beiden XQuery Engines auf *2600 MB* festgesetzt worden.

Die Messung des zur Auswertung einer Anfrage notwendigen Speicherbedarfs und die dazu benötigte Zeit wurde mit Hilfe eines *Monitoring-Skripts* realisiert. Dieses hat die entsprechenden Werte des Speicherbedarfs während der Auswertung einer Anfrage alle *0,025 Sekunden* direkt aus dem */proc-Dateisystem* ausgelesen, wobei stets der höchste Speicherbedarf einer jeden XMark-Anfrage auf ein XMark-Dokument für eine XQuery-Engine notiert wurde. Die *Auswertungszeit* ist von diesem Skript mit Hilfe der Systemzeit gemessen worden. Sie ist aus der Differenz der Systemzeiten zu Beginn und nach Beendigung der Auswertung einer XMark-Anfrage ermittelt worden. Die maximale Zeit, die eine XQuery Engine zur Auswertung einer XMark-Anfrage auf ein XMark-Dokument verwenden durfte, wurde auf *1800 Sekunden* festgesetzt. Bei Überschreiten dieser (Zeit-)Grenze wurde für die entsprechende XQuery Engine die Angabe „*timeout*“ vermerkt. XMark-Anfragen, die aufgrund eines zu kleinen Sprachfragments nicht auf einer XQuery Engine ausgeführt werden konnten, sind mit „*nicht verfügbar*“ (kurz: „*n.v.*“) gekennzeichnet worden. XQuery Engines, die bei der Auswertung einer XMark-Anfrage auf ein XMark-Dokument einen „*out of memory*“-Fehler verursacht haben, sind mit dem Kürzel „*oom*“ kenntlich gemacht worden.

Um die in dieser Arbeit hinzugefügten *Optimierungen* einschätzen zu können, wurden diese ebenfalls untersucht. Dabei sind die *Messungen* der *Auswertungszeit* und des *Speicherbedarf* einmal mit und einmal ohne Optimierungen erfolgt, wobei hier zu beachten ist, dass einige Optimierungen fest mit dem Kern der Anwendung verbunden sind und somit nicht ausgeschaltet werden konnten. So sind die drei Optimierungen „*Zuweisen von aggregierten Rollen*“ aus Abschnitt 3.4, „*Hashing der*

gespeicherten Elementnamen“ aus Abschnitt 3.4 und „Fixierung der Auswertungsreihenfolge der SignOff-Anweisungen“ aus Abschnitt 6.5 fest mit dem Kern der *GCX Engine Version 2.0* verbunden. Hingegen können die vier Optimierungen „Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden“ aus Abschnitt 6.1, „Entfernen von Tupeln in Abhängigkeiten bei enthaltenen Knotenmengen“ aus Abschnitt 6.2, „Entfernen von nicht benötigten Knoten eines Projektionsbaums“ aus Abschnitt 6.3 und „Entfernen von redundanten Rollen eines Projektionsbaums“ aus Abschnitt 6.4 mit Hilfe des „Deaktivierungs“-flags ausgeschaltet werden. Die noch verbleibende Optimierung zum „früherem Entfernen von gespeicherten Knoten“ ist aufgrund ihrer *Gegensätzlichkeit* zu einigen in dieser Arbeit hinzugefügten Optimierungen in einer eigenen Versuchsreihe untersucht worden.

Es sei an dieser Stelle nochmals angemerkt, dass keine als die zuvor in dieser Arbeit beschriebenen Veränderungen oder Optimierungen für die *GCX Engine Version 2.0* vorgenommen wurden oder auch keinerlei andere vorteilhafte Veränderungen oder Optimierungen für irgendeine XQuery Engine vollzogen worden sind.²

7.2 Experimentelle Ergebnisse

Die folgenden vier Tabellen enthalten die zur Auswertung der XMark-Anfragen auf die vier unterschiedlich großen XMark-Dokumente benötigten Zeiten und den dazu notwendigen Speicherbedarf als *Mittel* aus *drei Durchläufen* der einzelnen XQuery Engines. Es sei an dieser Stelle angemerkt, dass bei den bis zu 50 MB großen XMark-Dokumenten Messungsungenauigkeiten bei der Auswertungszeit und dem Speicherbedarf aufgetreten sind, womit die ermittelten Messwerte für einen Vergleich nur bedingt tauglich sind. Bei XMark-Dokumenten mit geringer Größe fällt die für die Auswertung der XMark-Anfrage benötigte Zeit und der dazu notwendige Speicherbedarf sehr gering aus, weshalb Unterschiede für die Messwerte mehr ins Gewicht fallen.

²Es sei an dieser Stelle angemerkt, dass die Dokumentprojektion in *GCX Engine Version 1.0b* nicht, wie in [20] bzw. [21] beschrieben, vorgenommen wird. Das Verwerfen von nicht benötigten Vorfahren eines relevanten Knotens ist nicht erfolgt.

7.2 EXPERIMENTELLE ERGEBNISSE

Engine Anfrage	1	2	3	4	5	6	7
XMark-Anfrage Q01	0,12 s 1,05 MB	0,17 s 1,13 MB	0,18 s 1,14 MB	0,17 s 1,14 MB	0,17 s 1,13 MB	1,18 s 55,88 MB	1,83 s 65,85 MB
XMark-Anfrage Q02	0,17 s 1,02 MB	0,58 s 1,09 MB	0,12 s 1,13 MB	0,16 s 1,13 MB	0,16 s 1,09 MB	1,39 s 60,05 MB	1,53 s 67,28 MB
XMark-Anfrage Q03	n.v.	n.v.	n.v.	0,25 s 1,14 MB	0,17 s 1,13 MB	1,62 s 70,05 MB	1,48 s 71,68 MB
XMark-Anfrage Q05	n.v.	n.v.	n.v.	0,09 s 1,13 MB	0,17 s 1,13 MB	1,22 s 60,28 MB	1,38 s 66,07 MB
XMark-Anfrage Q06	n.v.	n.v.	n.v.	0,42 s 1,67 MB	0,11 s 1,65 MB	1,18 s 51,33 MB	1,25 s 66,58 MB
XMark-Anfrage Q07	n.v.	n.v.	n.v.	0,11 s 1,90 MB	0,44 s 1,88 MB	1,26 s 62,36 MB	1,36 s 66,76 MB
XMark-Anfrage Q08	8,36 s 5,63 MB	15,10 s 5,99 MB	15,10 s 6,00 MB	11,43 s 5,99 MB	10,89 s 5,98 MB	1,71 s 61,98 MB	3,59 s 87,47 MB
XMark-Anfrage Q09	timeout	timeout	timeout	timeout	550,22 s 2,38 MB	1,46 s 59,06 MB	3,54 s 95,04 MB
XMark-Anfrage Q10	n.v.	n.v.	n.v.	229,93 s 3,75 MB	44,24 s 3,74 MB	10,13 s 333,09 MB	15,33 s 110,07 MB
XMark-Anfrage Q11	8,25 s 2,01 MB	14,32 s 1,97 MB	14,26 s 1,98 MB	6,88 s 1,97 MB	6,88 s 1,96 MB	7,76 s 212,55 MB	2,76 s 75,27 MB
XMark-Anfrage Q12	5,76 s 1,75 MB	8,97 s 1,69 MB	8,94 s 1,70 MB	3,71 s 1,68 MB	3,74 s 1,67 MB	5,29 s 190,07 MB	2,97 s 79,75 MB
XMark-Anfrage Q13	0,14 s 1,07 MB	0,13 s 1,17 MB	0,13 s 1,22 MB	0,15 s 1,21 MB	0,29 s 1,18 MB	0,75 s 55,86 MB	1,41 s 67,19 MB
XMark-Anfrage Q14	0,22 s 1,13 MB	0,22 s 1,84 MB	0,20 s 1,83 MB	0,16 s 1,79 MB	0,14 s 1,84 MB	1,47 s 70,85 MB	1,52 s 76,04 MB
XMark-Anfrage Q15	0,14 s 1,03 MB	0,14 s 1,12 MB	0,13 s 1,12 MB	0,13 s 1,15 MB	0,12 s 1,13 MB	1,17 s 51,23 MB	1,36 s 66,89 MB
XMark-Anfrage Q16	n.v.	n.v.	n.v.	0,16 s 1,15 MB	0,13 s 1,14 MB	1,59 s 54,64 MB	1,56 s 68,35 MB
XMark-Anfrage Q17	n.v.	n.v.	n.v.	0,24 s 1,13 MB	0,14 s 1,09 MB	0,95 s 59,93 MB	1,29 s 69,94 MB
XMark-Anfrage Q18	0,14 s 1,05 MB	0,41 s 1,09 MB	0,39 s 1,13 MB	0,24 s 1,13 MB	0,32 s 1,10 MB	1,17 s 56,01 MB	1,38 s 66,84 MB
XMark-Anfrage Q19	0,45 s 1,06 MB	0,11 s 1,48 MB	0,05 s 1,50 MB	0,10 s 1,50 MB	0,10 s 1,48 MB	1,64 s 53,33 MB	1,94 s 65,87 MB
XMark-Anfrage Q20	n.v.	n.v.	n.v.	0,42 s 1,64 MB	0,26 s 1,67 MB	1,01 s 55,78 MB	1,00 s 70,31 MB
1	GCX Engine Version 1.0b: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt)						
2	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt); ohne bzw. nur fest mit dem Kern der Anwendung verbundene Optimierungen						
3	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1; mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
4	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
5	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; ohne Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
6	Qizx/open Version 2.1: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)						
7	Saxon Version 9.1.0.1J: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)						

Tabelle 7.1: Auswertungszeiten und Speicherbedarf für 10 MB XMark-Dokument

KAPITEL 7 EXPERIMENTELLE RESULTATE

Engine \ Anfrage	1	2	3	4	5	6	7
XMark-Anfrage Q01	0,75 s 1,04 MB	0,65 s 1,13 MB	0,71 s 1,13 MB	0,69 s 1,14 MB	0,46 s 1,13 MB	2,60 s 196,97 MB	3,26 s 257,54 MB
XMark-Anfrage Q02	0,81 s 1,02 MB	0,80 s 1,09 MB	0,79 s 1,14 MB	0,74 s 1,14 MB	0,75 s 1,10 MB	3,65 s 237,67 MB	4,69 s 293,18 MB
XMark-Anfrage Q03	n.v.	n.v.	n.v.	0,81 s 1,15 MB	0,83 s 1,14 MB	4,12 s 233,69 MB	4,84 s 323,27 MB
XMark-Anfrage Q05	n.v.	n.v.	n.v.	0,57 s 1,13 MB	0,76 s 1,13 MB	2,39 s 191,06 MB	3,09 s 270,62 MB
XMark-Anfrage Q06	n.v.	n.v.	n.v.	0,73 s 1,67 MB	0,58 s 1,67 MB	2,51 s 193,70 MB	3,19 s 255,63 MB
XMark-Anfrage Q07	n.v.	n.v.	n.v.	0,92 s 2,99 MB	0,80 s 2,99 MB	2,93 s 215,15 MB	3,20 s 245,15 MB
XMark-Anfrage Q08	210,76 s 24,10 MB	383,84 s 25,23 MB	381,31 s 25,23 MB	286,44 s 25,20 MB	282,88 s 25,19 MB	3,64 s 225,58 MB	48,34 s 352,44 MB
XMark-Anfrage Q09	timeout	timeout	timeout	timeout	timeout	3,26 s 205,35 MB	63,04 s 375,58 MB
XMark-Anfrage Q10	n.v.	n.v.	n.v.	timeout	1059,45 s 13,93 MB	54,44 s 1159,32 MB	257,10 s 375,15 MB
XMark-Anfrage Q11	202,71 s 5,82 MB	353,54 s 5,28 MB	352,77 s 5,28 MB	166,74 s 5,27 MB	172,79 s 5,26 MB	180,84 s 1757,95 MB	32,05 s 270,09 MB
XMark-Anfrage Q12	136,27 s 4,47 MB	212,90 s 3,85 MB	211,95 s 3,85 MB	86,32 s 3,84 MB	86,46 s 3,83 MB	99,22 s 1759,05 MB	32,53 s 273,46 MB
XMark-Anfrage Q13	0,46 s 1,08 MB	0,61 s 1,18 MB	0,60 s 1,22 MB	0,59 s 1,22 MB	0,69 s 1,18 MB	2,77 s 209,43 MB	3,47 s 256,80 MB
XMark-Anfrage Q14	1,01 s 1,15 MB	1,02 s 1,87 MB	0,98 s 1,88 MB	0,96 s 1,87 MB	1,04 s 1,87 MB	6,43 s 243,25 MB	3,67 s 279,00 MB
XMark-Anfrage Q15	0,71 s 1,02 MB	0,93 s 1,12 MB	0,64 s 1,12 MB	0,82 s 1,17 MB	0,48 s 1,17 MB	2,51 s 209,80 MB	3,21 s 261,09 MB
XMark-Anfrage Q16	n.v.	n.v.	n.v.	0,43 s 1,15 MB	0,66 s 1,15 MB	2,54 s 189,10 MB	3,28 s 247,68 MB
XMark-Anfrage Q17	n.v.	n.v.	n.v.	0,67 s 1,14 MB	0,81 s 1,10 MB	2,70 s 187,15 MB	4,36 s 294,21 MB
XMark-Anfrage Q18	0,81 s 1,05 MB	0,41 s 1,09 MB	0,60 s 1,13 MB	0,72 s 1,14 MB	0,42 s 1,09 MB	2,47 s 208,24 MB	3,19 s 265,63 MB
XMark-Anfrage Q19	0,67 s 1,06 MB	0,76 s 1,49 MB	0,74 s 1,52 MB	0,48 s 1,52 MB	0,75 s 1,49 MB	2,94 s 214,96 MB	4,00 s 272,05 MB
XMark-Anfrage Q20	n.v.	n.v.	n.v.	0,69 s 3,30 MB	1,19 s 3,30 MB	2,69 s 190,41 MB	3,67 s 250,94 MB
1	GCX Engine Version 1.0b: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt)						
2	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt); ohne bzw. nur fest mit dem Kern der Anwendung verbundene Optimierungen						
3	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1; mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
4	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
5	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; ohne Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
6	Qizx/open Version 2.1: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)						
7	Saxon Version 9.1.0.1J: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)						

Tabelle 7.2: Auswertungszeiten und Speicherbedarf für 50 MB XMark-Dokument

7.2 EXPERIMENTELLE ERGEBNISSE

Engine Anfrage	1	2	3	4	5	6	7
XMark-Anfrage Q01	1,48 s 1,04 MB	1,52 s 1,13 MB	1,66 s 1,13 MB	1,68 s 1,14 MB	1,55 s 1,13 MB	4,11 s 308,95 MB	5,40 s 442,30 MB
XMark-Anfrage Q02	1,63 s 1,02 MB	1,53 s 1,09 MB	1,56 s 1,14 MB	1,31 s 1,14 MB	1,34 s 1,09 MB	5,67 s 373,63 MB	6,50 s 480,00 MB
XMark-Anfrage Q03	n.v.	n.v.	n.v.	1,49 s 1,15 MB	1,69 s 1,14 MB	5,30 s 347,46 MB	6,73 s 488,93 MB
XMark-Anfrage Q05	n.v.	n.v.	n.v.	1,14 s 1,13 MB	1,18 s 1,13 MB	4,03 s 314,75 MB	5,26 s 451,90 MB
XMark-Anfrage Q06	n.v.	n.v.	n.v.	1,45 s 1,67 MB	1,50 s 1,67 MB	4,20 s 310,98 MB	5,29 s 441,31 MB
XMark-Anfrage Q07	n.v.	n.v.	n.v.	1,84 s 4,26 MB	1,60 s 4,26 MB	4,25 s 319,85 MB	5,49 s 453,28 MB
XMark-Anfrage Q08	833,39 s 46,98 MB	1510,69 s 49,06 MB	1494,99 s 49,06 MB	1130,13 s 48,99 MB	1119,28 s 48,99 MB	6,25 s 363,60 MB	164,60 s 591,66 MB
XMark-Anfrage Q09	timeout	timeout	timeout	timeout	timeout	5,06 s 314,50 MB	226,51 s 642,14 MB
XMark-Anfrage Q10	n.v.	n.v.	n.v.	timeout	timeout	108,16 s 1924,82 MB	973,77 s 617,40 MB
XMark-Anfrage Q11	806,64 s 10,54 MB	1405,42 s 9,34 MB	1405,12 s 9,35 MB	651,18 s 9,29 MB	678,16 s 9,29 MB	oom	90,62 s 458,61 MB
XMark-Anfrage Q12	533,90 s 7,83 MB	835,52 s 6,45 MB	835,65 s 6,45 MB	342,17 s 6,44 MB	341,88 s 6,43 MB	oom	92,91 s 505,66 MB
XMark-Anfrage Q13	1,64 s 1,08 MB	0,87 s 1,18 MB	1,15 s 1,22 MB	1,27 s 1,22 MB	1,12 s 1,19 MB	4,45 s 315,30 MB	5,82 s 444,08 MB
XMark-Anfrage Q14	1,83 s 1,16 MB	2,02 s 1,87 MB	1,95 s 1,88 MB	1,85 s 1,87 MB	2,01 s 1,87 MB	18,87 s 524,28 MB	6,42 s 488,07 MB
XMark-Anfrage Q15	1,42 s 1,03 MB	1,31 s 1,12 MB	1,51 s 1,12 MB	1,18 s 1,21 MB	1,20 s 1,21 MB	3,94 s 320,47 MB	5,09 s 424,00 MB
XMark-Anfrage Q16	n.v.	n.v.	n.v.	1,55 s 1,15 MB	1,31 s 1,14 MB	3,91 s 297,65 MB	5,40 s 460,75 MB
XMark-Anfrage Q17	n.v.	n.v.	n.v.	1,26 s 1,13 MB	1,39 s 1,10 MB	4,39 s 323,45 MB	6,20 s 492,98 MB
XMark-Anfrage Q18	1,36 s 1,05 MB	1,29 s 1,09 MB	0,93 s 1,13 MB	0,96 s 1,14 MB	1,22 s 1,10 MB	4,09 s 309,78 MB	5,31 s 430,40 MB
XMark-Anfrage Q19	1,54 s 1,06 MB	1,46 s 1,49 MB	1,47 s 1,52 MB	1,60 s 1,52 MB	1,51 s 1,49 MB	4,48 s 326,66 MB	6,32 s 458,71 MB
XMark-Anfrage Q20	n.v.	n.v.	n.v.	1,36 s 5,32 MB	1,41 s 5,32 MB	4,22 s 317,53 MB	5,67 s 482,46 MB

1 GCX Engine Version 1.0b: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt)

2 GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt); ohne bzw. nur fest mit dem Kern der Anwendung verbundene Optimierungen

3 GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1; mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“

4 GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“

5 GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; ohne Optimierung zum „früherem Entfernen von gespeicherten Knoten“

6 Qizx/open Version 2.1: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)

7 Saxon Version 9.1.0.1J: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)

Tabelle 7.3: Auswertungszeiten und Speicherbedarf für 100 MB XMark-Dokument

KAPITEL 7 EXPERIMENTELLE RESULTATE

Engine \ Anfrage	1	2	3	4	5	6	7
XMark-Anfrage Q01	3,29 s 1,05 MB	2,87 s 1,13 MB	2,83 s 1,14 MB	2,73 s 1,14 MB	2,75 s 1,13 MB	7,18 s 554,92 MB	9,64 s 821,06 MB
XMark-Anfrage Q02	3,13 s 1,02 MB	3,22 s 1,09 MB	3,17 s 1,13 MB	2,91 s 1,14 MB	3,00 s 1,10 MB	9,06 s 620,79 MB	10,93 s 925,03 MB
XMark-Anfrage Q03	n.v.	n.v.	n.v.	3,29 s 1,15 MB	3,35 s 1,14 MB	9,40 s 615,43 MB	11,31 s 882,67 MB
XMark-Anfrage Q05	n.v.	n.v.	n.v.	2,78 s 1,13 MB	2,39 s 1,13 MB	7,02 s 570,12 MB	8,93 s 839,53 MB
XMark-Anfrage Q06	n.v.	n.v.	n.v.	2,22 s 1,70 MB	3,00 s 1,67 MB	7,26 s 553,35 MB	9,32 s 874,87 MB
XMark-Anfrage Q07	n.v.	n.v.	n.v.	3,15 s 6,77 MB	3,18 s 6,77 MB	7,66 s 554,27 MB	9,55 s 846,78 MB
XMark-Anfrage Q08	timeout	timeout	timeout	timeout	timeout	11,45 s 641,95 MB	704,34 s 955,23 MB
XMark-Anfrage Q09	timeout	timeout	timeout	timeout	timeout	8,14 s 573,09 MB	920,29 s 886,50 MB
XMark-Anfrage Q10	n.v.	n.v.	n.v.	timeout	timeout	oom	timeout
XMark-Anfrage Q11	timeout	timeout	timeout	timeout	timeout	oom	350,86 s 885,00 MB
XMark-Anfrage Q12	timeout	timeout	timeout	1365,96 s 11,75 MB	1370,31 s 11,74 MB	oom	360,50 s 846,17 MB
XMark-Anfrage Q13	5,54 s 1,07 MB	2,34 s 1,18 MB	1,96 s 1,22 MB	3,92 s 1,23 MB	3,66 s 1,18 MB	8,22 s 584,37 MB	13,20 s 835,92 MB
XMark-Anfrage Q14	4,21 s 1,16 MB	4,10 s 1,87 MB	3,92 s 1,88 MB	3,87 s 1,87 MB	4,10 s 1,87 MB	70,90 s 759,22 MB	10,56 s 868,31 MB
XMark-Anfrage Q15	2,85 s 1,03 MB	2,61 s 1,12 MB	2,68 s 1,12 MB	2,33 s 1,26 MB	2,58 s 1,26 MB	7,51 s 564,00 MB	9,49 s 832,69 MB
XMark-Anfrage Q16	n.v.	n.v.	n.v.	2,57 s 1,15 MB	2,13 s 1,14 MB	7,12 s 549,48 MB	9,76 s 848,55 MB
XMark-Anfrage Q17	n.v.	n.v.	n.v.	3,05 s 1,13 MB	2,76 s 1,10 MB	7,84 s 570,18 MB	10,96 s 910,14 MB
XMark-Anfrage Q18	2,84 s 1,05 MB	2,42 s 1,09 MB	2,22 s 1,13 MB	1,98 s 1,13 MB	2,68 s 1,09 MB	7,46 s 547,92 MB	9,52 s 824,76 MB
XMark-Anfrage Q19	2,95 s 1,06 MB	3,01 s 1,49 MB	2,94 s 1,52 MB	2,88 s 1,52 MB	2,93 s 1,49 MB	7,34 s 546,03 MB	10,84 s 855,60 MB
XMark-Anfrage Q20	n.v.	n.v.	n.v.	3,14 s 9,41 MB	2,64 s 9,41 MB	7,80 s 554,46 MB	10,26 s 865,61 MB
1	GCX Engine Version 1.0b: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt)						
2	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1 (Pfade mit einem Schritt); ohne bzw. nur fest mit dem Kern der Anwendung verbundene Optimierungen						
3	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.1; mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
4	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; mit Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
5	GCX Engine Version 2.0: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten); mit Optimierungen; ohne Optimierung zum „früherem Entfernen von gespeicherten Knoten“						
6	Qizx/open Version 2.1: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)						
7	Saxon Version 9.1.0.1J: XMark-Anfragen aus Unterabschnitt G.1.2 (Pfade mit mehreren Schritten)						

Tabelle 7.4: Auswertungszeiten und Speicherbedarf für 200 MB XMark-Dokument

7.3 Diskussion der Ergebnisse

Mit den einzelnen Versuchsreihen sollte primär die Minimierung des zur Auswertung einer (XMark-)Anfrage auf ein (XMark-)Dokument benötigten Speicherbedarfs der GCX Engine untersucht und mit anderen XQuery Engines verglichen werden. Da dabei die zur Auswertung einer XMark-Anfrage auf ein XMark-Dokument benötigte Zeit nicht außer Acht gelassen werden sollte, geben die Vergleiche für die GCX Engine auch Aufschluss über die erzielten Zeiten und der dafür verantwortlichen Elemente bzw. (System-)Komponenten.

Vergleich von Spalte eins mit Spalte zwei von Tabelle 7.1 bis Tabelle 7.4: Bei diesem Vergleich zwischen *GCX Engine Version 1.0b* und *GCX Engine Version 2.0*, deren Zeiten und Speicherbedarf in den ersten beiden Spalten zu finden sind, ist untersucht worden, inwieweit sich die *Erweiterungen* auf die zur Auswertung einer XMark-Anfrage auf ein XMark-Dokument benötigte *Zeit* und den dazu notwendigen *Speicherbedarf* positiv oder negativ ausgewirkt haben. Insgesamt, d.h. für alle XMark-Dokumente betrachtet, kann für die *Auswertungszeit* festgestellt werden, dass für XMark-Anfragen, die keine where-Klausel besitzen oder das Bilden von Joins nicht erfordern, diese *nahezu identisch* sind. Zwar ist die Mehrheit der XMark-Anfragen mit where-Klauseln versehen oder erfordern das Bilden von Joins, allerdings können für die verbleibenden XMark-Anfragen auf allen XMark-Dokumenten Unterschiede einzelner Auswertungszeiten auf *Messungsungenauigkeiten* zurückgeführt werden oder weisen nur eine geringe zeitliche Differenz auf. Für XMark-Anfragen, die *where-Klauseln* enthalten oder das Bilden von *Joins* erfordern, ist eine *Verschlechterung*, z.B. auffällig für die XMark-Anfragen Q08, Q11 und Q12, für alle XMark-Dokumente festzustellen. Ursache dafür sind die im Rahmen dieser Arbeit vorgenommenen *tiefgreifenden Veränderungen* in der (ursprünglichen) Implementierung des Ausgangspunktes, die insbesondere für where-Klauseln bzw. die für sie umgeschriebenen bedingten Ausdrücke und für Joins erfolgt sind. Neben diesen notwendigen Veränderungen für die *Erweiterungen* und *Optimierungen* ist zusätzlich ein *Vorteil* der *objektorientierten Programmierung*, *Codefragmente* und *Systemkomponenten* nicht erneut an verschiedenen Stellen zu reimplementieren, sondern direkt darauf zu „verweisen“ und dadurch darauf zugreifen zu können, aus-

genutzt worden. Dies hat allerdings auch viele interne Vergleiche und Prüfungen erfordert, die mehrmals während der Auswertung einer (XMark-) Anfrage erfolgen, weshalb der Großteil des Zeitverlusts darauf zurückzuführen ist. Dieser Vorteil der objektorientierten Programmierung ist in der (ursprünglichen) Implementierung des Ausgangspunktes nicht vollzogen worden, weshalb hier *GCX Engine Version 2.0* von den erreichten *Auswertungszeiten* von *GCX Engine Version 1.0b* abweicht und somit eine *allgemeine Verschlechterung* der Auswertungszeit für diese XMark-Anfragen zu konstatieren ist. Der *Speicherbedarf*, der zur Auswertung der XMark-Anfragen auf die XMark-Dokumente benötigt wird, ist in *GCX Engine Version 2.0* für alle XMark-Dokumente *leicht angestiegen*. Nur bei zwei XMark-Anfragen, Q11 und Q12, konnte der Speicherbedarf für alle XMark-Dokumente leicht reduziert werden. Der leichte Anstieg des Speicherbedarfs lässt sich, wie bereits für die where-Klauseln, mit verschiedenen *Veränderungen* begründen, bei denen wesentliche und tragende Systemkomponenten aufgrund der Erweiterung um Pfade mit mehreren Schritten reimplementiert wurden und das Allokieren von (weiterem neuem) Speicher erforderten. Dieser (weitere neu) allokierte Speicher ist allerdings konstant, d.h. nicht abhängig von der (XMark-)Dokumentgröße, sondern fester Speicher, der einmalig belegt wird. Der im Vergleich zur *GCX Engine Version 1.0b* benötigte geringere Speicherbedarf für die Auswertung der zwei XMark-Anfragen der *GCX Engine Version 2.0* ist im Wesentlichen auf zwei Ursachen zurückzuführen. Zum einen ist dafür die *Dokumentprojektion* verantwortlich, die in *GCX Engine Version 1.0b* für den Knotentest „text()“ wesentlich mehr Teile eines Dokuments bzw. seines Dokumentenbaums speichert als notwendig wären. Zum anderen sind dafür die in *GCX Engine Version 1.0b* eingefügten SignOff-Anweisungen in eine (XMark-) Anfrage verantwortlich, die bei ihrer Ausführung stets Teilbäume vollständig einlesen und somit dazu führen, dass ebenfalls wesentlich mehr gespeichert wird als notwendig wäre. Daher konnten aufgrund der „verbesserten“ Dokumentprojektion für die Behandlung des Knotentests „text()“ und durch eine leichte „Veränderung“ der Ausführung von SignOff-Anweisungen, so dass diese nicht ganze Teilbäume einlesen, für diese XMark-Anfragen in *GCX Engine Version 2.0* verglichen mit *GCX Engine Version 1.0b* für den *Speicherbedarf* *bessere Resultate* erzielt werden.

Vergleich von Spalte zwei mit Spalte drei von Tabelle 7.1 bis Tabelle 7.4: Der Vergleich zwischen *GCX Engine Version 2.0* und *GCX Engine Version 2.0* mit Optimierungen untersucht die im Rahmen dieser Arbeit *hinzugefügten Optimierungen* darauf, inwieweit sich die zur Auswertung einer XMark-Anfrage auf ein XMark-Dokument benötigte Zeit und der dazu notwendige Speicherbedarf durch sie verbessern lässt. Dabei konnten – global betrachtet – die zur Auswertung der XMark-Anfragen benötigten *Zeiten* für alle XMark-Dokumente *nicht verbessert* werden. Dass sich durch die in dieser Arbeit hinzugefügten Optimierungen keine erkennbaren Verbesserungen erzielen ließen, liegt an der in Abschnitt 6.6 beschriebenen Tatsache der *Gegensätzlichkeit* einiger hinzugefügter Optimierungen zur Optimierung zum „früherem Entfernen von gespeicherten Knoten“. Da diese für *GCX Engine Version 2.0* in Spalte drei aktiviert ist, sind im Prinzip die *beiden verglichenen Engines nahezu identisch*, da fast keine der in dieser Arbeit hinzugefügten Optimierungen – außer im Wesentlichen der Optimierung zum „früherem Entfernen von gespeicherten Knoten“ – Anwendung fand, d.h. diese Optimierung hat fast alle weiteren neu hinzugefügten Optimierungen außer Kraft gesetzt. Eine *Reduzierung* des *Speicherbedarfs*, der sich für die *GCX Engine Version 2.0* in Spalte drei aufgrund der aktivierten Optimierung zum „früherem Entfernen von gespeicherten Knoten“ im Gegensatz zur *GCX Engine Version 2.0* in Spalte zwei ergeben sollte, ist für *keine XMark-Anfrage* auf die XMark-Dokumente *erkennbar*. Deswegen wird im nächsten Vergleich die auftretende *Gegensätzlichkeit* einiger hinzugefügter Optimierungen zu der Optimierung zum „früherem Entfernen von gespeicherten Knoten“ dieses Vergleichs nochmals näher untersucht.

Vergleich von Spalte vier mit Spalte fünf von Tabelle 7.1 bis Tabelle 7.4: In diesem Vergleich der *GCX Engine Version 2.0* mit Optimierungen und derselben ohne die Optimierung zum „früherem Entfernen von gespeicherten Knoten“ wurde untersucht, inwieweit sich die in Abschnitt 6.6 beschriebene *Gegensätzlichkeit* auf die Auswertung der XMark-Anfragen auf die XMark-Dokumente auswirkt. Mit anderen Worten: Können die in dieser Arbeit hinzugefügten Optimierungen eine Verbesserung in der Auswertungszeit und dem Speicherbedarf gegenüber der Optimierung zum „früherem Entfernen von gespeicherten Knoten“ erzielen? Beim Vergleich dieser beiden Spalten ist für die *Auswertungszeiten* insgesamt festzustellen, dass

die zur Auswertung der XMark-Anfragen benötigten Zeiten *nahezu identisch* sind bzw. für *GCX Engine Version 2.0* in Spalte fünf *leicht verringert* werden konnten. Da aufgrund der Optimierungen lediglich konstante Zeit verloren geht, ist es aufgrund von Messungsungenauigkeiten nicht auszuschließen, dass ein wesentlich größerer Vorteil bzgl. der Auswertungszeiten auf Seiten der *GCX Engine Version 2.0* ohne die Optimierung zum „früherem Entfernen von gespeicherten Knoten“ liegt. Dieser Vorteil ist z.B. für die XMark-Anfrage Q09 auf das 10 MB XMark-Dokument und für die XMark-Anfrage Q10 auf das 50 MB XMark-Dokument so *signifikant*, dass ein „timeout“, der für *GCX Engine Version 2.0* in Spalte vier aufgetreten ist, verhindert werden konnte. Hierbei gilt es besonders zu beachten, dass die zur Auswertung einer XMark-Anfrage maximal zur Verfügung stehende Zeit auf *1800 Sekunden* festgesetzt worden ist und diese Grenze für diese beiden XMark-Anfragen auf die jeweiligen XMark-Dokumente *weit unterschritten* wird. Der Grund der zeitlichen Verbesserung von *GCX Engine Version 2.0* in Spalte fünf liegt im Wesentlichen an dem Entfernen von Knoten³ und an dem Entfernen von redundanten Rollen von Knoten des einer XMark-Anfrage zugehörigen Projektionsbaums. Hierdurch werden weniger Rollen den zu speichernden Knoten vergeben, weniger SignOff-Anweisungen in die XMark-Anfragen eingefügt und somit die Speicherbereinigung weniger oft aufgerufen. Der Speicherbedarf ist für *GCX Engine Version 2.0* ohne die Optimierung zum „früherem Entfernen von gespeicherten Knoten“ entweder *identisch* oder *geringer*, was das Potential der in dieser Arbeit hinzugefügten Optimierungen untermauert. Das Entfernen von Knoten und das Entfernen von redundanten Rollen von Knoten des einer (XMark-)Anfrage zugehörigen Projektionsbaums äußert sich in weniger Rollen, die den zu speichernden Knoten vergeben werden, weniger in eine (XMark-)Anfrage eingefügten SignOff-Anweisungen und somit in weniger Ausführungen der Speicherbereinigung, was letztlich weniger für den zur Auswertung einer (XMark-)Anfrage notwendigen Speicherbedarf bedeutet,

³Zum Entfernen von Knoten eines Projektionsbaums sind an dieser Stelle alle im Rahmen dieser Arbeit hinzugefügten Optimierungen zu verstehen, die Knoten direkt aus dem Projektionsbaum entfernen oder dies indirekt tun. So werden durch die zwei Optimierungen „Entfernen von Tupeln in Abhängigkeiten bei identischen Pfaden“ und „Entfernen von Tupeln in Abhängigkeiten bei enthaltenen Knotenmengen“ die der Konzipierung der Optimierung entsprechenden Tupel aus den Abhängigkeiten entfernt und somit indirekt verhindert, dass diese als Knoten in einem Projektionsbaum eingefügt werden.

als die Optimierung zum „früherem Entfernen von gespeicherten Knoten“ liefert.

Vergleich von Spalte fünf mit Spalten sechs und sieben von Tabelle 7.1 bis Tabelle 7.4: Im letzten Vergleich werden *GCX Engine Version 2.0* mit *Qizx/open Version 2.1* und *Saxon Version 9.1.0.1J* gegenübergestellt. Dabei kann festgestellt werden, dass *GCX Engine Version 2.0* in Spalte fünf in allen XMark-Anfragen, die weder eine where-Klausel oder das Bilden von Joins erfordern, eine schnellere Auswertungszeit besitzt, als die in den Versuchsreihen zum Vergleich herangezogenen XQuery Engines *Qizx/open Version 2.1* und *Saxon Version 9.1.0.1J*. Nur in den fünf XMark-Anfragen Q08, Q09, Q10, Q11 und Q12, die entweder eine where-Klausel enthalten oder das Bilden von Joins erfordern, muss die *GCX Engine Version 2.0* diesen beiden den Vortritt lassen. Hierfür ursächlich sind die bereits erwähnten tiefgreifenden Veränderungen in den Systemkomponenten der where-Klauseln aber auch allgemein die verwendete Join-Strategie. Diese Strategie basiert in *GCX Engine Version 1.0b* und auch in *GCX Engine Version 2.0* auf (naive) nested loop joins, die nicht konkurrenzfähig ist. Der Speicherbedarf, dessen Minimierung das primäre Ziel der GCX Engine ist, liegt für *alle* XMark-Anfragen auf allen XMark-Dokumenten *deutlich unter den erhaltenen Messwerten* von *Qizx/open Version 2.1* und *Saxon Version 9.1.0.1J* und verdeutlicht den Benefit der aktiven Speicherbereinigung. Selbst XMark-Anfragen, deren Auswertung aufgrund der Überschreitung der Zeitgrenze nicht möglich war, lassen erwarten, dass der Speicherbedarf gegenüber *Qizx/open Version 2.1* und *Saxon Version 9.1.0.1J* weit unter diesen liegen dürfte, da bei den beiden Java basierten XQuery Engines mit der Dokumentgröße – u.a. wegen der automatischen Speicherverwaltung von Java und der JVM – auch der Speicherbedarf ansteigt.

Der Vollständigkeit halber sei hier, wenn auch nur am Rande und aufgrund „unterschiedlicher“ XMark-Anfragen nur bedingt aussagekräftig, auf den *Vergleich von Spalte drei mit Spalte vier von Tabelle 7.1 bis Tabelle 7.4* eingegangen. Für die zur Auswertung der XMark-Anfragen benötigten *Zeiten* kann insgesamt festgehalten werden, dass diese *nahezu identisch* sind oder für *GCX Engine Version 2.0* in Spalte vier für XMark-Anfragen, die eine where-Klausel enthalten oder das Bilden von Joins erfordern, *signifikant verbessert* werden konnten. Ursache hierfür sind die in

den where-Klauseln der XMark-Anfragen verwendeten Pfade mit mehreren Schritten, die im Vergleich zu den XMark-Anfragen mit Pfaden mit einem Schritt für *GCX Engine Version 1.0b* in Spalte drei, eine schnellere Auswertungszeit besitzen. Auch der *Speicherbedarf* ist für alle XMark-Anfragen auf allen XMark-Dokumenten *nahezu identisch*, da die in den beiden Spalten jeweilig verwendeten XMark-Anfragen zum einen äquivalent sind und zum anderen die Speicherung der zur Auswertung der XMark-Anfragen notwendigen Knoten eines Dokumentenbaums nicht von Pfaden mit mehreren Schritten abhängt.

Als Fazit für *GCX Engine Version 2.0* kann aus den durchgeführten Experimenten festgestellt werden, dass der Speicherbedarf bei wachsender Dokumentgröße annähernd konstant ist und die Auswertungszeiten linear ansteigend zur Dokumentgröße sind.

Kapitel 8

Zusammenfassung und Ausblick

In dieser Arbeit ist die aktive Speicherbereinigung und deren Implementierung in der GCX Engine um Pfade mit mehreren Schritten und um Aggregatfunktionen erweitert worden. Dabei sind nicht nur die Standard-Aggregatfunktionen, wie sie auch im XQuery-Standard zu finden sind, eingefügt worden, sondern auch eine Reihe weiterer, deren Einsatz für statistische Zwecke geeignet sind. Neben diesen Erweiterungen sind auch eine Reihe weiterer Optimierungen hinzugefügt und eine bestehende Optimierung erweitert worden. Des Weiteren sind aufgrund der in dieser Arbeit erfolgten Erweiterungen und Optimierungen tiefgreifende Veränderungen in der (ursprünglichen) Implementierung des Ausgangspunktes erfolgt und dabei tragende Systemkomponenten umgeschrieben worden. Das Resultat der in dieser Arbeit erfolgten Implementierungen hat sich in der aktuell finalen Versionsnummer 2.0 niedergeschlagen, was die Weiterentwicklung der GCX Engine im Rahmen dieser Arbeit deutlich macht.

Die durchgeführten Experimente, bei der die GCX Engine mit zwei bekannten Vertretern ihrer Art – Qizx/open und Saxon – verglichen wurde, haben gezeigt, dass das primäre Ziel, die *Minimierung* des *Speicherbedarfs*, durch die Erweiterung um Pfade mit mehreren Schritten und auch mit der Unterstützung von Aggregatfunktionen *erreicht* wird. Die einzelnen Versuchsreihen haben bewiesen, dass die GCX Engine Version 2.0 deutlich weniger Speicherbedarf benötigt als andere XQuery Engines. Dies resultiert aus der effizienten Umsetzung der Dokumentprojektion innerhalb der GCX Engine. Aber auch die aktive Speicherbereinigung – im Besonderen die dynamische Analyse, die nicht mehr benötigte Teile des Dokuments bzw. seines Dokumentenbaums so früh als möglich aus dem Speicher entfernt, um den Speicherbedarf stets minimal zu halten – trägt wesentlich dazu bei.

Neben dem primären Ziel der Reduzierung des Speicherbedarfs sollte aber auch die *Auswertungszeit* einer Anfrage nicht außer Acht gelassen werden. Hier haben die Experimente gezeigt, dass die GCX Engine für einen Großteil der Anfragen *sehr gute Resultate* liefert bzw. um das 2- bis 4-fache *schneller* als vergleichbare XQuery Engines ist. *Problematisch* aus Sicht der zur Auswertung einer Anfrage benötigten Zeit erwiesen sich allerdings Anfragen, die eine *where-Klausel* enthalten oder das Bilden von *Joins* erfordern. Die Verschlechterung der Auswertungszeit von Anfragen, die eine where-Klausel besitzen, ist auf die im Rahmen dieser Arbeit vorgenommen tiefgreifenden *Veränderungen* von Systemkomponenten zurückzuführen. Anfragen, die Joins enthalten, sind bereits mit der dieser Arbeit zugrunde liegenden Implementierung nicht in einer akzeptablen Zeit auszuwerten gewesen. Für solche Anfragen sind weitere traditionelle Optimierungen, wie z.B. *hash joins*, notwendig, um die GCX Engine kompetitiv zu machen.

Die in den Experimenten ebenfalls untersuchten *Optimierungen* haben gezeigt, dass diese sich auf die zur Auswertung einer Anfrage benötigten *Zeit* und den dazu notwendigen *Speicherbedarf* niederschlagen können. Bezüglich der Optimierung zum „früherem Entfernen von gespeicherten Knoten“ kann die Aussage getroffen werden, dass durch ihre Gegensätzlichkeit zu einigen in dieser Arbeit hinzugefügten Optimierungen, abhängig von der Anfrage und dem Dokument, sowohl Zeit als auch Speicherbedarf bei der Auswertung einer Anfrage verloren gehen können. Aufgrund dessen ist es eher sinnvoll, auf diese Optimierung zu verzichten und stattdessen nur die in dieser Arbeit hinzugefügten Optimierungen zu verwenden.

Für einen Ausblick wären die bereits in [20] genannten Erweiterungen und Neuerungen zu nennen. Dazu zählen Standard-Techniken, wie sie auch in Datenbanken verwendet werden, um bspw. statt der bisher in der GCX Engine verwendeten Join-Strategie, (naive) nested loop joins, die Implementierung *erweiterter Join-Strategien*, wie z.B. *hash joins* oder *Index basierte Joins*, zur Auswertung einer Anfrage zu nutzen. Weiter könnten Erweiterungen und Optimierungen, die, wie in [2, 6, 31] beschrieben, auf Schema Wissen beruhen, eingesetzt werden. *Schema Wissen*, das aus einer *DTD* oder aus *XML-Schema* stammt, könnte helfen, die Auswertungszeit und auch den Speicherbedarf zu reduzieren. Für die diesen Ideen zugrunde liegenden Überlegungen und Beispiele sei an dieser Stelle auf die Originalarbeit in [20] verwiesen.

Eine weitere nennenswerte Erweiterung wäre die *Unterstützung* von *Attributen* innerhalb der Elemente bzw. innerhalb deren öffnender Tags in einer Anfrage. Aber auch die Unterstützung von Attributen, die aus den Elementen bzw. deren öffnender Tags eines Dokuments stammen, in Form von Attributknoten eines Dokumentenbaums wäre denkbar. Es wäre auch möglich, das *Sprachfragment XQ* um weitere XQuery-Funktionen, wie z.B. „*distinct-values*“, zu erweitern. Diese Funktion ist angesichts der in dieser Arbeit hinzugefügten Aggregatfunktionen nun dahingehend plausibel, als sie mehrfach vorkommende Werte, sprich Duplikate, nicht in das Ergebnis einer Aggregatfunktion einrechnet und eine neue interessante Klasse praxisrelevanter Anfragen ermöglicht. Schließlich wäre über eine, wenn auch der objektorientierten Programmierung widersprechenden, *Reimplementierung* der *Codefragmente* nachzudenken, um den Zeitverlust, der aufgrund der where-Klausel auftritt, wieder wett zumachen, um somit wieder die Zeiten zu erhalten, die annähernd der dieser Arbeit zugrunde liegenden Implementierung in der GCX Engine Version 1.0b entsprechen.

Abbildungsverzeichnis

3.1 XQuery Sprachfragment XQ	21
3.2 Inferenzregeln zum „Hineindrücken“ von bedingten Ausdrücken in FR- Ausdrücke	33
3.3 Inferenzregeln zum Einfügen von SignOff-Anweisungen in eine Anfrage .	37
3.4 GCX Systemarchitektur	44
4.1 XQuery Sprachfragment XQ'	56
5.1 XQuery Sprachfragment XQ''	70

Tabellenverzeichnis

2.1 XPath-Achsen und lokalisierte Knotenmengen	13
7.1 Auswertungszeiten und Speicherbedarf für 10 MB XMark-Dokument . .	115
7.2 Auswertungszeiten und Speicherbedarf für 50 MB XMark-Dokument . .	116
7.3 Auswertungszeiten und Speicherbedarf für 100 MB XMark-Dokument .	117
7.4 Auswertungszeiten und Speicherbedarf für 200 MB XMark-Dokument .	118
B.1 Pfade als Beispiele für XPath-Achsen	148
B.1 Pfade als Beispiele für XPath-Achsen (Fortsetzung)	149
B.1 Pfade als Beispiele für XPath-Achsen (Fortsetzung)	150
B.1 Pfade als Beispiele für XPath-Achsen (Fortsetzung)	151
B.1 Pfade als Beispiele für XPath-Achsen (Fortsetzung)	152
B.1 Pfade als Beispiele für XPath-Achsen (Fortsetzung)	153
C.1 Beispiel der aktiven Speicherbereinigung	155
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	156
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	157
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	158
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	159
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	160
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	161
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	162
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	163
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	164
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	165
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	166
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	167

C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	168
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	169
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	170
C.1 Beispiel der aktiven Speicherbereinigung (Fortsetzung)	171

Quelltextverzeichnis

3.1	Algorithmus $signOffInsert_Q(\$x)$ zum Einfügen von SignOff-Anweisungen	39
3.2	Algorithmus $signOff(\$x/\pi, r)$ zum Ausführen einer SignOff-Anweisung	42
B.1	Beispiel eines XML-Dokuments für eine Bibliothek	147
D.1	Erweitertes XML-Dokument für eine Bibliothek	173
E.1	XML-Dokument für eine Personenliste von Entleiher	175
F.1	XML-Dokument von entliehenen Büchern und deren Entleiher	177
G.1	XMark-Anfrage Q01 mit Pfaden mit einem Schritt	179
G.2	XMark-Anfrage Q02 mit Pfaden mit einem Schritt	179
G.3	XMark-Anfrage Q08 mit Pfaden mit einem Schritt	180
G.4	XMark-Anfrage Q09 mit Pfaden mit einem Schritt	180
G.5	XMark-Anfrage Q11 mit Pfaden mit einem Schritt	181
G.6	XMark-Anfrage Q12 mit Pfaden mit einem Schritt	181
G.7	XMark-Anfrage Q13 mit Pfaden mit einem Schritt	182
G.8	XMark-Anfrage Q14 mit Pfaden mit einem Schritt	182
G.9	XMark-Anfrage Q15 mit Pfaden mit einem Schritt	182
G.10	XMark-Anfrage Q18 mit Pfaden mit einem Schritt	183
G.11	XMark-Anfrage Q19 mit Pfaden mit einem Schritt	183
G.12	XMark-Anfrage Q01 mit Pfaden mit mehreren Schritten	184
G.13	XMark-Anfrage Q02 mit Pfaden mit mehreren Schritten	184
G.14	XMark-Anfrage Q03 mit Pfaden mit mehreren Schritten	184
G.15	XMark-Anfrage Q05 mit Pfaden mit mehreren Schritten	185
G.16	XMark-Anfrage Q06 mit Pfaden mit mehreren Schritten	185
G.17	XMark-Anfrage Q07 mit Pfaden mit mehreren Schritten	185

G.18 XMark-Anfrage Q08 mit Pfaden mit mehreren Schritten	185
G.19 XMark-Anfrage Q09 mit Pfaden mit mehreren Schritten	186
G.20 XMark-Anfrage Q10 mit Pfaden mit mehreren Schritten	186
G.21 XMark-Anfrage Q11 mit Pfaden mit mehreren Schritten	187
G.22 XMark-Anfrage Q12 mit Pfaden mit mehreren Schritten	187
G.23 XMark-Anfrage Q13 mit Pfaden mit mehreren Schritten	187
G.24 XMark-Anfrage Q14 mit Pfaden mit mehreren Schritten	188
G.25 XMark-Anfrage Q15 mit Pfaden mit mehreren Schritten	188
G.26 XMark-Anfrage Q16 mit Pfaden mit mehreren Schritten	188
G.27 XMark-Anfrage Q17 mit Pfaden mit mehreren Schritten	188
G.28 XMark-Anfrage Q18 mit Pfaden mit mehreren Schritten	189
G.29 XMark-Anfrage Q19 mit Pfaden mit mehreren Schritten	189
G.30 XMark-Anfrage Q20 mit Pfaden mit mehreren Schritten	189

Abkürzungsverzeichnis

API	A pplication P rogramming I nterface
bspw.	b eispielsweise
bzw.	b eziehungsweise
ca.	c irca
d.h.	d as h eißt
DFA	D eterministic F inite A utomaton
DOM	D ocument O bject M odel
DTD	D ocument T ype D efinition
etc.	E t c etera
GCX	G arbage C ollected X Query
JRE	J ava R untime E nvironment
JVM	J ava V irtual M achine
MB	M egabyte
n.v.	n icht v erfügbar
oom	o ut o f m emory
PCDATA	P arsed C haracter D ata
s	S ekunde/-n
s.	siehe
SAX	S imple A PI for X ML
u.a.	u nter a nderem
u.U.	u nter U mständen
XMark	X ML B ench M ark
XML	E xtensible M arkup L anguage
XPath	X ML P ath L anguage
XQuery	X ML Q uery L anguage
z.B.	zum B eispiel

Literaturverzeichnis

- [1] Amélie Marian, Jérôme Siméon: *Projecting XML Documents*.
In: *VLDB*, Seiten 213–224, 2003.
<http://www.vldb.org/conf/2003/papers/S08P01.pdf>, letzter Abruf:
11. 10. 2008.
2, 22, 24, 25
- [2] Bertram Ludäscher, Pratik Mukhopadhyay, Yannis Papakonstantinou: *A Transducer-Based XML Query Processor*.
In: *VLDB*, Seiten 227–238, 2002.
<http://daks.ucdavis.edu/~ludaesch/Paper/xsm-vldb02.pdf>, letzter Abruf:
11. 10. 2008.
126
- [3] Christoph Koch: *On The Complexity Of Nonrecursive XQuery And Functional Query Languages On Complex Values*.
ACM Transactions On Database System, 31(4):1215–1256, 2006.
22, 23
- [4] Christoph Koch, Stefanie Scherzinger, Michael Schmidt: *User Manual GCX – Garbage Collected XQuery*, 2007.
http://dbis.informatik.uni-freiburg.de/content/projects/GCX/download/gcxman_1-0beta.pdf, letzter Abruf: 11. 10. 2008.
21
- [5] Christoph Koch, Stefanie Scherzinger, Nicole Schweikardt, Bernhard Stegmaier: *FluXQuery: An Optimizing XQuery Processor For Streaming XML Data*.
In: *VLDB*, Seiten 1309–1312, 2004.
<http://www.cs.cornell.edu/~koch/www.infosys.uni-sb.de/home/scherzin/documents/demo.pdf>, letzter Abruf: 11. 10. 2008.

111

- [6] Christoph Koch, Stefanie Scherzinger, Nicole Schweikardt, Bernhard Stegmaier: *Schema-based Scheduling Of Event Processors And Buffer Minimization For Queries On Structured Data Streams*.
In: *VLDB*, Seiten 228–239, 2004.
<http://www.vldb.org/conf/2004/RS6P2.PDF>, letzter Abruf: 11.10.2008.
22, 111, 126
- [7] Christoph Koch, Stefanie Scherzinger, Nicole Schweikardt, Bernhard Stegmaier: *The FluXQuery Engine*, 2004.
<http://www.cs.cornell.edu/~koch/www.infosys.uni-sb.de/home/scherzin/FluXQuery.html>, letzter Abruf: 11.10.2008.
111
- [8] CWI: *MonetDB/XQuery*, 2008.
<http://monetdb.cwi.nl/XQuery/>, letzter Abruf: 11.10.2008.
112
- [9] Dan Olteanu, Holger Meuss, Tim Furche, François Bry: *XPath: Looking Forward*.
In: *EDBT Workshops*, Seiten 109–127, 2002.
<http://www.cis.uni-muenchen.de/people/Meuss/Pub/XMLDM02.ps.gz>.
14
- [10] Georg Lausen: *XML-Technologie: Grundlagen*, 2007.
http://download.informatik.uni-freiburg.de/lectures/DB2/2007SS/Slides/03_01_XMLTechnologien.pdf, letzter Abruf: 11.10.2008.
7
- [11] Georg Lausen: *XML-Technologie: XPath*, 2007.
http://download.informatik.uni-freiburg.de/lectures/DB2/2007SS/Slides/03_03_XMLTechnologien.pdf, letzter Abruf: 11.10.2008.
7, 13
- [12] Georg Lausen: *XML-Technologie: XQuery*, 2007.

- http://download.informatik.uni-freiburg.de/lectures/DB2/2007SS/Slides/04_01_XQuery.pdf, letzter Abruf: 11. 10. 2008.
7, 16
- [13] Georg Lausen: *XML-Technologie: XQuery*, 2007.
http://download.informatik.uni-freiburg.de/lectures/DB2/2007SS/Slides/04_02_XQuery.pdf, letzter Abruf: 11. 10. 2008.
7
- [14] Gerome Miklau, Dan Suciu: *Containment And Equivalence For An XPath Fragment*.
In: *PODS*, Seiten 65–76, 2002.
<http://www.cs.umass.edu/~miklau/pubs/pods2002MS/p65-miklau.pdf>.
86, 87, 88, 89
- [15] Gerome Miklau, Dan Suciu: *Containment And Equivalence For A Fragment Of XPath*.
J. ACM, 51(1):2–45, 2004.
<http://www.cs.umass.edu/~miklau/pubs/jacm2004/xpath-jacm2004.pdf>.
86, 87, 88, 89
- [16] Markus Mauch: *XQuery-Anfragen*, 2006.
<http://wwwipd.ira.uka.de/~oosem/S2D2/material/6-Mauch.pdf>.
2
- [17] Mary Fernández, Jérôme Siméon, Cindy Chen, Byron Choi, Vladimir Gapeyev, Amélie Marian, Philippe Michiels, Nicola Onose, Douglas Petkanics, Christopher Rath, Christopher Ré, Michael Stark, Gargi Sur, Avinash Vyas, Philip Wadler: *The Galax System – "The XQuery Implementation For Discriminating Hackers"*, 2007.
<http://www.galaxquery.org/>, letzter Abruf: 11. 10. 2008.
111
- [18] Meike Klettke, Holger Meyer: *XML & Datenbanken – Konzepte, Sprachen und Systeme*.

- Dpunkt Verlag, 1. Auflage, 2003.
<http://www.xml-und-datenbanken.de/>, letzter Abruf: 11. 10. 2008.
2
- [19] Michael Kay: *SAXON – The XSLT And XQuery Processor*, 2007.
<http://saxon.sourceforge.net/>, letzter Abruf: 11. 10. 2008.
111
- [20] Michael Schmidt, Stefanie Scherzinger, Christoph Koch: *Combined Static And Dynamic Analysis For Effective Buffer Minimization In Streaming XQuery Evaluation*.
Diplomarbeit, Universität des Saarlandes – Lehrstuhl für Informationssysteme, 2006.
http://www.informatik.uni-freiburg.de/~mschmidt/docs/thesis_csada.pdf.
1, 2, 4, 19, 20, 47, 66, 111, 114, 126
- [21] Michael Schmidt, Stefanie Scherzinger, Christoph Koch: *Combined Static And Dynamic Analysis For Effective Buffer Minimization In Streaming XQuery Evaluation*.
In: *ICDE*, Seiten 236–245, 2007.
<http://www.informatik.uni-freiburg.de/~mschmidt/docs/csada.pdf>.
2, 4, 19, 20, 33, 37, 39, 42, 44, 47, 66, 111, 114
- [22] Paul R. Wilson: *Uniprocessor Garbage Collection Techniques*.
In: *IWMM*, Seiten 1–42, 1992.
<ftp://ftp.cs.utexas.edu/pub/garbage/gcsurvey.ps>, letzter Abruf:
11. 10. 2008.
31
- [23] Ralph Busse, Mike Carey, Daniela Florescu, Martin Kersten, Ioana Manolescu, Albrecht Schmidt, Florian Waas: *XMark – An XML Benchmark Project*, 2008.
<http://www.xml-benchmark.org/>, letzter Abruf: 11. 10. 2008.
109, 110

- [24] SQL Anywhere Server – SQL-Referenzhandbuch: *Mathematische Formeln für die Aggregatfunktionen*, 2007.
http://www.ianywhere.com/developer/product_manuals/sqlanywhere/1001/de/html/dbugde10/ug-ug-olap-s-6410540.html, letzter Abruf: 11. 10. 2008.
73
- [25] SQL Anywhere Server – SQL-Referenzhandbuch: *STDDEV_POP-Funktion [Aggregat]*, 2007.
http://www.ianywhere.com/developer/product_manuals/sqlanywhere/1001/de/html/dbrfde10/rf-functions-s-5578387.html, letzter Abruf: 11. 10. 2008.
- [26] SQL Anywhere Server – SQL-Referenzhandbuch: *STDDEV_SAMP-Funktion [Aggregat]*, 2007.
http://www.ianywhere.com/developer/product_manuals/sqlanywhere/1001/de/html/dbrfde10/rf-functions-s-5578387a.html, letzter Abruf: 11. 10. 2008.
- [27] SQL Anywhere Server – SQL-Referenzhandbuch: *VAR_POP-Funktion [Aggregat]*, 2007.
http://www.ianywhere.com/developer/product_manuals/sqlanywhere/1001/de/html/dbrfde10/rf-functions-s-5578389.html, letzter Abruf: 11. 10. 2008.
- [28] SQL Anywhere Server – SQL-Referenzhandbuch: *VAR_SAMP-Funktion [Aggregat]*, 2007.
http://www.ianywhere.com/developer/product_manuals/sqlanywhere/1001/de/html/dbrfde10/rf-functions-s-5578387b.html, letzter Abruf: 11. 10. 2008.
73
- [29] Stéphane Bressan, Barbara Catania, Zoé Lacroix, Ying Guang Li, Anna Maddalena: *Accelerating Queries By Pruning XML Documents*.
Data Knowledge Engineering, 54(2):211–240, 2005.

- <http://www.dcs.bbk.ac.uk/~klee08/papers/bressan-pruning-xml-dke.pdf>.
2, 24
- [30] Todd J. Green, Gerome Miklau, Makoto Onizuka, Dan Suci: *Processing XML Streams With Deterministic Automata*.
In: *ICDT*, Seiten 173–189, 2003.
<http://www.cs.umass.edu/~miklau/OldWebpage/pubs/icdt2003GMOS/streamingXML-DFA.pdf>, letzter Abruf: 11. 10. 2008.
46
- [31] Véronique Benzaken, Giuseppe Castagna, Dario Colazzo, Kim Nguyen: *Type-Based XML Projection*.
In: *VLDB*, Seiten 271–282, 2006.
<http://www.vldb.org/conf/2006/p271-benzaken.pdf>, letzter Abruf: 11. 10. 2008.
2, 24, 25, 126
- [32] (W3C), World Wide Web Consortium: *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. *W3C Recommendation, 16 August 2006*, 2006.
<http://www.w3.org/TR/2006/REC-xml-20060816/>, letzter Abruf: 11. 10. 2008.
7, 8
- [33] World Wide Web Consortium (W3C): *XML Path Language (XPath) Version 1.0*. *W3C Recommendation, 16 November 1999*, 1999.
<http://www.w3.org/TR/1999/REC-xpath-19991116>, letzter Abruf: 11. 10. 2008.
7, 11, 15
- [34] World Wide Web Consortium (W3C): *XML Path Language (XPath) 2.0*. *W3C Recommendation, 23 January 2007*, 2007.
<http://www.w3.org/TR/2007/REC-xpath20-20070123/>, letzter Abruf: 11. 10. 2008.
7, 11, 14, 15

- [35] World Wide Web Consortium (W3C): *XML Schema Part 2: Datatypes Second Edition*. *W3C Recommendation*, 28 October 2004, 2007.
<http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/>, letzter Abruf:
11. 10. 2008.
71, 73
- [36] World Wide Web Consortium (W3C): *XQuery 1.0: An XML Query Language*.
W3C Recommendation, 23 January 2007, 2007.
<http://www.w3.org/TR/2007/REC-xquery-20070123/>, letzter Abruf:
11. 10. 2008.
7, 9, 15, 16, 17
- [37] World Wide Web Consortium (W3C): *XQuery 1.0 and XPath 2.0 Formal Semantics*.
W3C Recommendation, 23 January 2007, 2007.
<http://www.w3.org/TR/2007/REC-xquery-semantics-20070123/>, letzter
Abruf: 11. 10. 2008.
17, 23
- [38] World Wide Web Consortium (W3C): *XQuery 1.0 and XPath 2.0 Functions and Operators*.
W3C Recommendation, 23 January 2007, 2007.
<http://www.w3.org/TR/2007/REC-xpath-functions-20070123/>, letzter
Abruf: 11. 10. 2008.
71, 72, 80
- [39] XMLmind: *Qizx/open*, 2008.
<http://www.xmlmind.com/qizx/qizxopen.shtml>, letzter Abruf:
11. 10. 2008.
111

Anhang A

Übersicht der beiliegenden CD

A.1 Ordnerstruktur und Ordnerbeschreibungen

/	
├── ausarbeitung.....	Dieser Ordner enthält im pdf-Format die Ausarbeitung dieser Arbeit.
├── implementierungen.....	Dieser Ordner enthält alle Implementierungen, die im Rahmen dieser Arbeit angefertigt wurden.
│ ├── dbis_cpp_gcx.....	(C ⁺⁺ -)Implementierungen der (Erweiterung und Optimierung der) „Garbage Collected XQuery“ (GCX) Engine.
├── literatur.....	Dieser Ordner enthält die aus dem Internet verwendete Literatur dieser Arbeit.
└── sonstiges.....	Dieser Ordner enthält weitere Dokumente, z.B. Cover der beiliegenden CD, XML-Dokumente, XQuery-Anfragen, Messergebnisse der Experimente etc..

Anhang B

Extensible Markup Language (XML)

B.1 Beispiel XML-Dokument

```
<bib>
  <book>
    <year>1994</year>
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book>
    <year>1992</year>
    <title>Advanced Programming In The Unix Environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  </book>
  <book>
    <year>2000</year>
    <title>Data On The Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
  </book>
  <book>
    <year>1999</year>
    <title>The Economics Of Technology And Content For Digital TV</title>
    <editor><last>Gerbarg</last><first>Darcy</first>
      <affiliation>CITI</affiliation></editor>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
  </book>
</bib>
```

Quelltext B.1: Beispiel eines XML-Dokuments für eine Bibliothek

B.2 Beispiele XPath-Achsen

Tabelle B.1: Pfade als Beispiele für XPath-Achsen

<i>ACHSE: child</i>
<i>Pfad:</i> /descendant::author/child::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1["n1:bib"] --- n2["n2:book"] n2 --- n3["n3:year"] n2 --- n5["n5:title"] n2 --- n7["n7:author"] n2 --- n12["n12:publisher"] n2 --- n14["n14:price"] n3 --- n4["n4:'1994'"] n5 --- n6["n6:'TCP/IP Illustrated'"] n7 --- n8["n8:last"] n7 --- n10["n10:first"] n8 --- n9["n9:'Stevens'"] n10 --- n11["n11:'W.'"] n12 --- n13["n13:'Addison-Wesley'"] n14 --- n15["n15:'65.95'"] </pre>
<i>ACHSE: descendant</i>
<i>Pfad:</i> /descendant::author/descendant::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1["n1:bib"] --- n2["n2:book"] n2 --- n3["n3:year"] n2 --- n5["n5:title"] n2 --- n7["n7:author"] n2 --- n12["n12:publisher"] n2 --- n14["n14:price"] n3 --- n4["n4:'1994'"] n5 --- n6["n6:'TCP/IP Illustrated'"] n7 --- n8["n8:last"] n7 --- n10["n10:first"] n8 --- n9["n9:'Stevens'"] n10 --- n11["n11:'W.'"] n12 --- n13["n13:'Addison-Wesley'"] n14 --- n15["n15:'65.95'"] </pre>

Fortsetzung auf der nächsten Seite ...

Tabelle B.1: Pfade als Beispiele für XPath-Achsen (Fortsetzung)

Fortsetzung von vorheriger Seite ...

<i>ACHSE: descendant-or-self</i>
<i>Pfad:</i> /descendant::author/descendant-or-self::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1["n1:bib"] --- n2["n2:book"] n2 --- n3["n3:year"] n2 --- n5["n5:title"] n2 --- n7["n7:author"] n2 --- n12["n12:publisher"] n2 --- n14["n14:price"] n3 --- n4["n4:'1994'"] n5 --- n6["n6:'TCP/IP Illustrated'"] n7 --- n8["n8:last"] n7 --- n10["n10:first"] n8 --- n9["n9:'Stevens'"] n10 --- n11["n11:'W.'"] n12 --- n13["n13:'Addison-Wesley'"] n14 --- n15["n15:'65.95'"] </pre>
<i>ACHSE: parent</i>
<i>Pfad:</i> /descendant::author/parent::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1["n1:bib"] --- n2["n2:book"] n2 --- n3["n3:year"] n2 --- n5["n5:title"] n2 --- n7["n7:author"] n2 --- n12["n12:publisher"] n2 --- n14["n14:price"] n3 --- n4["n4:'1994'"] n5 --- n6["n6:'TCP/IP Illustrated'"] n7 --- n8["n8:last"] n7 --- n10["n10:first"] n8 --- n9["n9:'Stevens'"] n10 --- n11["n11:'W.'"] n12 --- n13["n13:'Addison-Wesley'"] n14 --- n15["n15:'65.95'"] style n2 stroke-width:2px </pre>
<i>Fortsetzung auf der nächsten Seite ...</i>

Tabelle B.1: Pfade als Beispiele für XPath-Achsen (Fortsetzung)

Fortsetzung von vorheriger Seite ...

<i>ACHSE: ancestor</i>	
<i>Pfad:</i> /descendant::author/ancestor::node()	
<i>Lokalisierte Knotenmenge:</i>	
<pre> graph TD n1["n1:bib"] --> n2["n2:book"] n2 --> n3["n3:year"] n2 --> n5["n5:title"] n2 --> n7["n7:author"] n2 --> n12["n12:publisher"] n2 --> n14["n14:price"] n3 --> n4["n4:'1994'"] n5 --> n6["n6:'TCP/IP Illustrated'"] n7 --> n8["n8:last"] n7 --> n10["n10:first"] n8 --> n9["n9:'Stevens'"] n10 --> n11["n11:'W.'"] n12 --> n13["n13:'Addison-Wesley'"] n14 --> n15["n15:'65.95'"] </pre>	
<i>ACHSE: ancestor-or-self</i>	
<i>Pfad:</i> /descendant::author/ancestor-or-self::node()	
<i>Lokalisierte Knotenmenge:</i>	
<pre> graph TD n1["n1:bib"] --> n2["n2:book"] n2 --> n3["n3:year"] n2 --> n5["n5:title"] n2 --> n7["n7:author"] n2 --> n12["n12:publisher"] n2 --> n14["n14:price"] n3 --> n4["n4:'1994'"] n5 --> n6["n6:'TCP/IP Illustrated'"] n7 --> n8["n8:last"] n7 --> n10["n10:first"] n8 --> n9["n9:'Stevens'"] n10 --> n11["n11:'W.'"] n12 --> n13["n13:'Addison-Wesley'"] n14 --> n15["n15:'65.95'"] </pre>	

Fortsetzung auf der nächsten Seite ...

Tabelle B.1: Pfade als Beispiele für XPath-Achsen (Fortsetzung)

Fortsetzung von vorheriger Seite ...

<i>ACHSE: following</i>
<i>Pfad:</i> /descendant::author/following::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1[n1:bib] --> n2[n2:book] n2 --> n3[n3:year] n2 --> n5[n5:title] n2 --> n7[n7:author] n2 --> n12[n12:publisher] n2 --> n14[n14:price] n3 --> n4[n4:"1994"] n5 --> n6[n6:"TCP/IP Illustrated"] n7 --> n8[n8:last] n7 --> n10[n10:first] n8 --> n9[n9:"Stevens"] n10 --> n11[n11:"W."] n12 --> n13[n13:"Addison-Wesley"] n14 --> n15[n15:"65.95"] </pre>
<i>ACHSE: following-sibling</i>
<i>Pfad:</i> /descendant::author/following-sibling::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1[n1:bib] --> n2[n2:book] n2 --> n3[n3:year] n2 --> n5[n5:title] n2 --> n7[n7:author] n2 --> n12[n12:publisher] n2 --> n14[n14:price] n3 --> n4[n4:"1994"] n5 --> n6[n6:"TCP/IP Illustrated"] n7 --> n8[n8:last] n7 --> n10[n10:first] n8 --> n9[n9:"Stevens"] n10 --> n11[n11:"W."] n12 --> n13[n13:"Addison-Wesley"] n14 --> n15[n15:"65.95"] </pre>

Fortsetzung auf der nächsten Seite ...

Tabelle B.1: Pfade als Beispiele für XPath-Achsen (Fortsetzung)

Fortsetzung von vorheriger Seite ...

<i>ACHSE: preceding</i>
<i>Pfad:</i> /descendant::author/preceding::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1["n1:bib"] --- n2["n2:book"] n2 --- n3["n3:year"] n2 --- n5["n5:title"] n2 --- n7["n7:author"] n2 --- n12["n12:publisher"] n2 --- n14["n14:price"] n3 --- n4["n4:'1994'"] n5 --- n6["n6:'TCP/IP Illustrated'"] n7 --- n8["n8:last"] n7 --- n10["n10:first"] n8 --- n9["n9:'Stevens'"] n10 --- n11["n11:'W.'"] n12 --- n13["n13:'Addison-Wesley'"] n14 --- n15["n15:'65.95'"] </pre>
<i>ACHSE: preceding-sibling</i>
<i>Pfad:</i> /descendant::author/preceding-sibling::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1["n1:bib"] --- n2["n2:book"] n2 --- n3["n3:year"] n2 --- n5["n5:title"] n2 --- n7["n7:author"] n2 --- n12["n12:publisher"] n2 --- n14["n14:price"] n3 --- n4["n4:'1994'"] n5 --- n6["n6:'TCP/IP Illustrated'"] n7 --- n8["n8:last"] n7 --- n10["n10:first"] n8 --- n9["n9:'Stevens'"] n10 --- n11["n11:'W.'"] n12 --- n13["n13:'Addison-Wesley'"] n14 --- n15["n15:'65.95'"] </pre>

Fortsetzung auf der nächsten Seite ...

Tabelle B.1: Pfade als Beispiele für XPath-Achsen (Fortsetzung)

Fortsetzung von vorheriger Seite ...

ACHSE: <i>self</i>
<i>Pfad:</i> /descendant::author/self::node()
<i>Lokalisierte Knotenmenge:</i>
<pre> graph TD n1["n1:bib"] --- n2["n2:book"] n2 --- n3["n3:year"] n2 --- n5["n5:title"] n2 --- n7["n7:author"] n2 --- n12["n12:publisher"] n2 --- n14["n14:price"] n3 --- n4["n4:1994"] n5 --- n6["n6:TCP/IP Illustrated"] n7 --- n8["n8:last"] n7 --- n10["n10:first"] n8 --- n9["n9:Stevens"] n10 --- n11["n11:W."] n12 --- n13["n13:Addison-Wesley"] n14 --- n15["n15:65.95"] style n7 stroke-width:2px </pre>

Anhang C

Aktive Speicherbereinigung: Garbage Collected XQuery (GCX)

C.1 Beispiel der aktiven Speicherbereinigung

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17

Dieses Beispiel verdeutlicht Schritt für Schritt die Auswertung einer Anfrage auf ein Dokument aus Sicht der einzelnen <i>Schnittstellen</i> der GCX Engine. Dabei sei das Dokument aus Beispiel 2.1 als Datenstrom erhalten und die darauf auszuführende Anfrage aus Beispiel 3.2 gegeben. Aus der abgeschlossenen statischen Analyse ist der Projektionsbaum mit Rollen in Beispiel 3.11 und die in Beispiel 3.17 umgeschriebene Anfrage mit SignOff-Anweisungen, die vom <i>Anfrageevaluator</i> strikt sequenziell auf den sich (später) im Speicher befindlichen noch zu projizierenden Dokumentenbaum ausgeführt wird, erhalten worden. Innerhalb dieses Beispiels wird mit <i>for_{\$z}</i> die for-Klausel bezeichnet, die die Variable <i>\$z</i> einführt.	
SCHRITT 1	
<i>Eingabedatenstrom:</i>	
<i>Ausgabedatenstrom:</i>	<r>
<i>Speicherinhalt:</i>	∅

Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

<i>Fortsetzung von vorheriger Seite ...</i>	
Der <i>Anfrageevaluator</i> gibt den öffnenden Tag $\langle r \rangle$ aus und versucht, die $for_{\$bib}$ -Klausel zu evaluieren. Da sich allerdings zu diesem Zeitpunkt noch keine Knoten im Speicher ¹ befinden (\emptyset), muss der <i>Anfrageevaluator</i> auf Eingabe seitens des <i>Dokumentprojektors</i> warten.	
SCHRITT 2	
<i>Eingabedatenstrom:</i> $\langle bib \rangle$	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	$bib\{r_0\}$
Der <i>Dokumentprojektor</i> liest den öffnenden Tag $\langle bib \rangle$, dessen Elementknoten im Dokumentenbaum in der Knotenmenge, die durch den in Knoten n_2 beschriebenen Pfad ² des Projektionsbaums lokalisiert wird, enthalten ist. Aus diesem Grund wird der bib-Elementknoten des Dokumentenbaums mit Rolle r_0 versehen und gespeichert. ³ Der <i>Anfrageevaluator</i> evaluiert die $for_{\$bib}$ -Klausel und bindet die Variable $\$bib$ an den gespeicherten bib-Knoten. Als nächstes wertet er den Körper der return-Klausel bzw. den nach ihr stehenden Ausdruck aus. Dabei versucht er, die $for_{\$x}$ -Klausel zu evaluieren, muss allerdings aufgrund der fehlenden Knoten im Speicher auf neue Eingabe seitens des <i>Dokumentprojektors</i> warten.	
<i>Fortsetzung auf der nächsten Seite ...</i>	

¹Der Speicher ist, sowohl ohne ein Dokument erhalten zu haben bzw. zu Beginn der Auswertung einer Anfrage, als auch nach Beendigung der Auswertung, grundsätzlich niemals leer. Er enthält stets die Dokumentwurzel, an die die Variable $\$root$ (einmalig) fest gebunden wird. Die Ausgabe der Dokumentwurzel wird unterbunden und dient als Elternknoten für das/die vom Datenstrom stammende(n) Wurzelement(e), die aufgrund des Datenstrom-Szenarios auch mehrere sein können, falls diese von mehreren nacheinander folgenden Dokumenten stammen.

²Der (beschriebene) Pfad eines Knotens n des Projektionsbaums ergibt sich aus der Konkatination der einzelnen Pfade bzw. Schritte von der Wurzel „/“ nach n zu einem Pfad.

³Zusätzlich erhalten alle gespeicherten Knoten, deren schließender Tag ihres Elements noch nicht gelesen wurde, ein „unfinished“-flag, welches, sobald der schließende Tag ihres Elements gelesen wird, wieder entfernt wird. Falls ein Knoten, der ein „unfinished“-flag besitzt, zwischenzeitlich gelöscht werden soll, erfolgt dessen Löschung unmittelbar nach Entfernen des „unfinished“-flags, d.h. nach dem Lesen des schließenden Tags ihres Elements.

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

SCHRITT 3	
<i>Eingabedatenstrom:</i>	<book>
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	$\begin{array}{c} \text{bib}\{r_0\} \\ \\ \text{book}\{r_1, r_3, r_4\} \end{array}$
<p>Der <i>Dokumentprojektor</i> liest den öffnenden Tag <book>, dessen Elementknoten im Dokumentenbaum in den Knotenmengen, die durch die in Knoten n_3, n_5 und n_6 beschriebenen Pfade des Projektionsbaums lokalisiert werden, enthalten ist. Aus diesem Grund wird der book-Elementknoten des Dokumentenbaums mit den Rollen r_1, r_3 und r_4 versehen und gespeichert. Der <i>Anfrageevaluator</i> evaluiert die <i>for</i>_{$\\$x$}-Klausel und bindet die Variable $\\$x$ an den gespeicherten book-Knoten. Als nächstes wertet er den Körper der return-Klausel bzw. den nach ihr stehenden Ausdruck aus. Dabei versucht er, den bedingten Ausdruck „if (not(exists($\\$x/price$))) then $\\$x$ else ()“ auszuwerten, muss allerdings aufgrund der fehlenden Knoten im Speicher eine neue Eingabe vom <i>Dokumentprojektor</i> anfordern.</p>	
SCHRITT 4	
<i>Eingabedatenstrom:</i>	<year>1994</year>
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	$\begin{array}{c} \text{bib}\{r_0\} \\ \\ \text{book}\{r_1, r_3, r_4\} \\ \\ \text{year}\{r_3\} \\ \\ \text{"1994"} \end{array}$

Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

Fortsetzung von vorheriger Seite ...

Der *Dokumentprojektor* liest den öffnenden Tag `<year>`, den Elementtext „1994“ und den schließenden Tag `</year>` in dieser Reihenfolge.⁴ Der Elementknoten im Dokumentenbaum ist in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten. Aus diesem Grund wird der `year`-Elementknoten des Dokumentenbaums mit Rolle r_3 versehen und gespeichert. Der diesem Elementknoten zugehörige Textknoten im Dokumentenbaum ist ebenfalls in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten. Daher wird auch der diesem Elementknoten zugehörige Textknoten des Dokumentenbaums gespeichert, erhält allerdings bei seiner Speicherung keine Rolle zugewiesen.⁵ Da weiterhin die Knoten für die Auswertung des bedingten Ausdrucks „`if (not(exists($x/price))) then $x else ()`“ nicht vorliegen, wartet der *Anfrageevaluator* wieder auf neue Eingabe.

Fortsetzung auf der nächsten Seite ...

⁴In diesem Beispiel werden Elemente mit Elementtext in einem Schritt behandelt. Dies entspricht nicht dem eigentlichen Vorgehen der GCX Engine, die dies in mehreren Schritten der Abfolge „Lesen des öffnenden Tags vom Dokumentprojektor → Speichermanager → Anfrageevaluator → ‚Nachschlagen‘ im Speicher durch den Speichermanager → Lesen des Elementtexts vom Dokumentprojektor → Speichermanager → Anfrageevaluator → ‚Nachschlagen‘ im Speicher durch den Speichermanager → Lesen des schließenden Tags vom Dokumentprojektor → Speichermanager → Anfrageevaluator“ vollziehen würde.

⁵Elementtext wird gespeichert, falls sein Textknoten im Dokumentenbaum in den Knotenmengen, die durch die in einem oder mehreren Knoten beschriebenen Pfade des Projektionsbaums lokalisiert werden, enthalten ist. Das Entfernen dieser Knoten aus dem Speicher erfolgt zusammen mit dem Entfernen des (Element-)Knotens, dem sie zugehörig sind.

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

SCHRITT 5
<i>Eingabedatenstrom:</i> <title>TCP/IP Illustrated</title>
<i>Ausgabedatenstrom:</i>
<i>Speicherinhalt:</i> <div style="text-align: center; margin: 10px 0;"> <pre> graph TD bib["bib{r0}"] --> book["book{r1,r3,r4}"] book --> year["year{r3}"] book --> title["title{r3,r5}"] year --> year_val["1994"] title --> title_val["TCP/IP Illustrated"] </pre> </div>
<p>Der <i>Dokumentprojektor</i> liest den öffnenden Tag <title>, den Elementtext „TCP/IP Illustrated“ und den schließenden Tag </title> in dieser Reihenfolge. Der Elementknoten im Dokumentenbaum ist in den Knotenmengen, die durch die in Knoten n_5 und n_7 beschriebenen Pfade des Projektionsbaums lokalisiert werden, enthalten. Aus diesem Grund wird der title-Elementknoten des Dokumentenbaums mit den Rollen r_3 und r_5 versehen und gespeichert. Der diesem Elementknoten zugehörige Textknoten im Dokumentenbaum ist ebenfalls in den Knotenmengen, die durch die in Knoten n_5 bzw. n_7 beschriebenen Pfade des Projektionsbaums lokalisiert werden, enthalten. Daher wird auch der diesem Elementknoten zugehörige Textknoten des Dokumentenbaums gespeichert, erhält allerdings bei seiner Speicherung keine Rolle zugewiesen. Da weiterhin die Knoten für die Auswertung des bedingten Ausdrucks „if (not(exists($\\$x/price$))) then $\\$x$ else ()“ nicht vorliegen, wartet der <i>Anfrageevaluator</i> wieder auf neue Eingabe.</p>

Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

SCHRITT 6	
<i>Eingabedatenstrom:</i> <author>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	<pre> graph TD bib["bib{r0}"] --- book["book{r1,r3,r4}"] book --- year["year{r3}"] book --- title["title{r3,r5}"] book --- author["author{r3}"] year --- year_val["1994"] title --- title_val["TCP/IP Illustrated"] </pre>
<p>Der <i>Dokumentprojektor</i> liest den öffnenden Tag <author>, dessen Elementknoten im Dokumentenbaum in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten ist. Aus diesem Grund wird der author-Elementknoten des Dokumentenbaums mit Rolle r_3 versehen und gespeichert. Da weiterhin die Knoten für die Auswertung des bedingten Ausdrucks „if (not(exists(\$x/price))) then \$x else ()“ nicht vorliegen, wartet der <i>Anfrageevaluator</i> wieder auf neue Eingabe.</p>	
<i>Fortsetzung auf der nächsten Seite ...</i>	

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

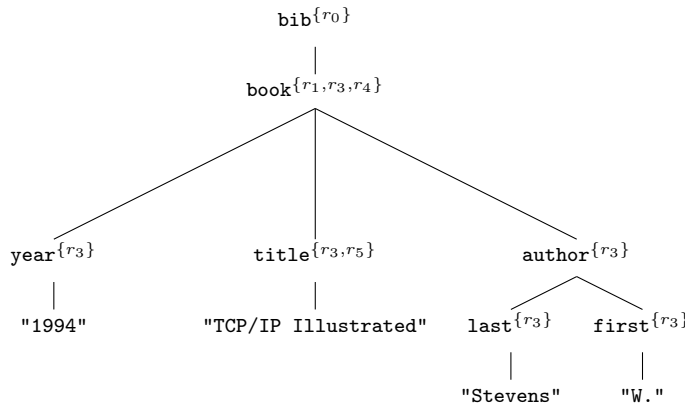
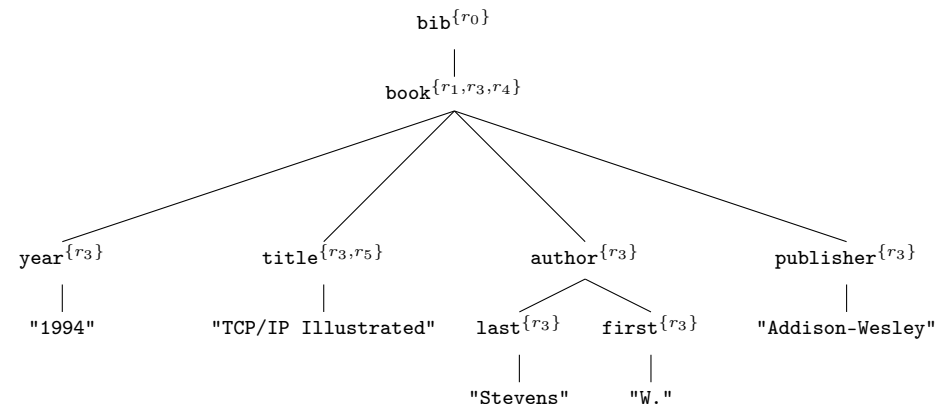
SCHRITT 7
<i>Eingabedatenstrom:</i> <last>Stevens</last>
<i>Ausgabedatenstrom:</i>
<p><i>Speicherinhalt:</i></p> <pre> bib{r0} book{r1,r3,r4} / \ year{r3} title{r3,r5} author{r3} "1994" "TCP/IP Illustrated" last{r3} "Stevens" </pre>
<p>Der <i>Dokumentprojektor</i> liest den öffnenden Tag <last>, den Elementtext „Stevens“ und den schließenden Tag </last> in dieser Reihenfolge. Der Elementknoten im Dokumentenbaum ist in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten. Aus diesem Grund wird der last-Elementknoten des Dokumentenbaums mit Rolle r_3 versehen und gespeichert. Der diesem Elementknoten zugehörige Textknoten im Dokumentenbaum ist ebenfalls in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten. Daher wird auch der diesem Elementknoten zugehörige Textknoten des Dokumentenbaums gespeichert, erhält allerdings bei seiner Speicherung keine Rolle zugewiesen. Da weiterhin die Knoten für die Auswertung des bedingten Ausdrucks „if (not(exists(\$x/price))) then \$x else ()“ nicht vorliegen, wartet der <i>Anfrageevaluator</i> wieder auf neue Eingabe.</p>

Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

SCHRITT 8
<i>Eingabedatenstrom:</i> <first>W.</first>
<i>Ausgabedatenstrom:</i>
<p><i>Speicherinhalt:</i></p> <pre> graph TD bib["bib{r0}"] --- book["book{r1,r3,r4}"] book --- year["year{r3}"] book --- title["title{r3,r5}"] book --- author["author{r3}"] year --- year_val["1994"] title --- title_val["TCP/IP Illustrated"] author --- last["last{r3}"] author --- first["first{r3}"] last --- last_val["Stevens"] first --- first_val["W."] </pre>
<p>Der <i>Dokumentprojektor</i> liest den öffnenden Tag <first>, den Elementtext „W.“ und den schließenden Tag </first> in dieser Reihenfolge. Der Elementknoten im Dokumentenbaum ist in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten. Aus diesem Grund wird der first-Elementknoten des Dokumentenbaums mit Rolle r_3 versehen und gespeichert. Der diesem Elementknoten zugehörige Textknoten im Dokumentenbaum ist ebenfalls in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten. Daher wird auch der diesem Elementknoten zugehörige Textknoten des Dokumentenbaums gespeichert, erhält allerdings bei seiner Speicherung keine Rolle zugewiesen. Da weiterhin die Knoten für die Auswertung des bedingten Ausdrucks „if (not(exists(\$x/price))) then \$x else ()“ nicht vorliegen, wartet der <i>Anfrageevaluator</i> wieder auf neue Eingabe.</p>
<i>Fortsetzung auf der nächsten Seite ...</i>

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

SCHRITT 9	
<i>Eingabedatenstrom:</i> </author>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	 <pre> graph TD bib["bib{r0}"] --- book["book{r1,r3,r4}"] book --- year["year{r3}"] book --- title["title{r3,r5}"] book --- author["author{r3}"] year --- year_val["1994"] title --- title_val["TCP/IP Illustrated"] author --- last["last{r3}"] author --- first["first{r3}"] last --- last_val["Stevens"] first --- first_val["W."] </pre>
Der <i>Dokumentprojektor</i> liest den schließenden Tag </author>. ⁶	
SCHRITT 10	
<i>Eingabedatenstrom:</i> <publisher>Addison-Wesley</publisher>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	 <pre> graph TD bib["bib{r0}"] --- book["book{r1,r3,r4}"] book --- year["year{r3}"] book --- title["title{r3,r5}"] book --- author["author{r3}"] book --- publisher["publisher{r3}"] year --- year_val["1994"] title --- title_val["TCP/IP Illustrated"] author --- last["last{r3}"] author --- first["first{r3}"] publisher --- publisher_val["Addison-Wesley"] last --- last_val["Stevens"] first --- first_val["W."] </pre>

Fortsetzung auf der nächsten Seite ...

⁶An dieser Stelle wird bspw. das am gespeicherten author-Knoten versehene „unfinished“-flag wieder entfernt.

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

Fortsetzung von vorheriger Seite ...

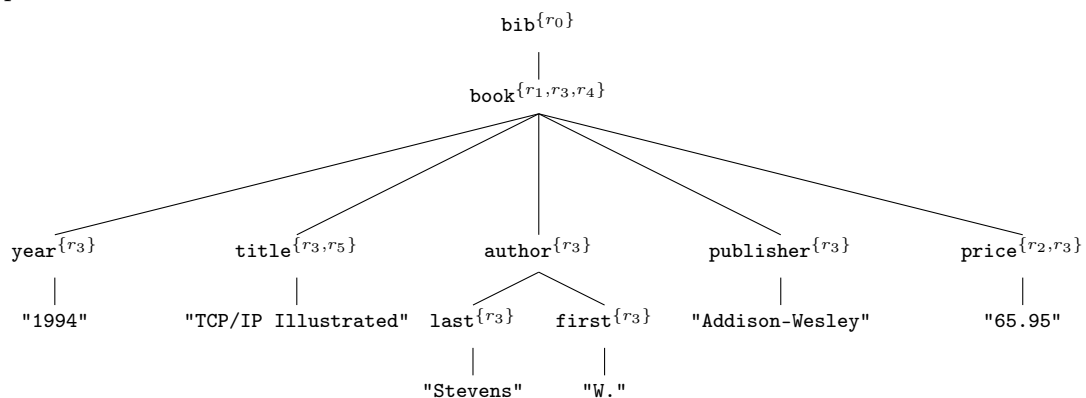
Der *Dokumentprojektor* liest den öffnenden Tag `<publisher>`, den Elementtext „Addison-Wesley“ und den schließenden Tag `</publisher>` in dieser Reihenfolge. Der Elementknoten im Dokumentenbaum ist in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaums lokalisiert wird, enthalten. Aus diesem Grund wird der `publisher`-Elementknoten des Dokumentenbaums mit Rolle r_3 versehen und gespeichert. Der diesem Elementknoten zugehörige Textknoten im Dokumentenbaum ist ebenfalls in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaum lokalisiert wird, enthalten. Daher wird auch der diesem Elementknoten zugehörige Textknoten des Dokumentenbaums gespeichert, erhält allerdings bei seiner Speicherung keine Rolle zugewiesen. Da weiterhin die Knoten für die Auswertung des bedingten Ausdrucks „if (not(exists(\$x/price))) then \$x else ()“ nicht vorliegen, wartet der *Anfrageevaluator* wieder auf neue Eingabe.

SCHRITT 11

Eingabedatenstrom: `<price>65.95</price>`

Ausgabedatenstrom:

Speicherinhalt:



Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

Fortsetzung von vorheriger Seite ...

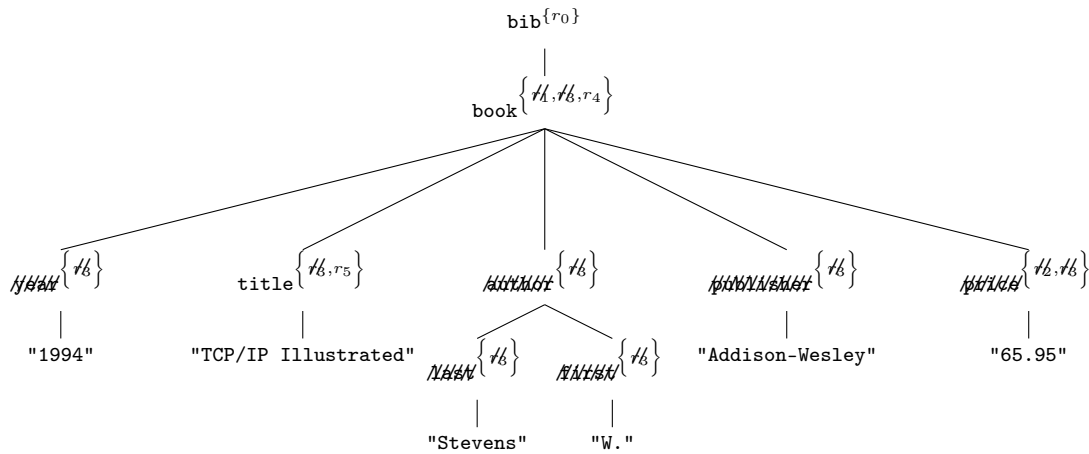
Der *Dokumentprojektor* liest den öffnenden Tag `<price>`, den Elementtext „65.95“ und den schließenden Tag `</price>` in dieser Reihenfolge. Der Elementknoten im Dokumentenbaum ist in den Knotenmengen, die durch die in Knoten n_4 und n_5 beschriebenen Pfade des Projektionsbaums lokalisiert werden, enthalten. Aus diesem Grund wird der price-Elementknoten des Dokumentenbaums mit den Rollen r_2 und r_3 versehen und gespeichert. Der diesem Elementknoten zugehörige Textknoten im Dokumentenbaum ist in der Knotenmenge, die durch den in Knoten n_5 beschriebenen Pfad des Projektionsbaum lokalisiert wird, enthalten. Daher wird auch der diesem Elementknoten zugehörige Textknoten des Dokumentenbaums gespeichert, erhält allerdings bei seiner Speicherung keine Rolle zugewiesen.⁷

SCHRITT 12

Eingabedatenstrom:

Ausgabedatenstrom:

Speicherinhalt:



Fortsetzung auf der nächsten Seite ...

⁷Die Auswertung des bedingten Ausdrucks „if (not(exists(\$x/price))) then \$x else ()“ kann theoretisch bereits vom Anfrageevaluator erfolgen, nachdem der Dokumentprojektor den öffnenden Tag `<price>` gelesen hat. Dies wird aber aufgrund der im Anschluss folgenden auszuführenden SignOff-Anweisungen aufgeschoben bis das „unfinished“-flag am gespeicherten price-Knoten entfernt wird, d.h. der schließende Tag `</price>` gelesen wird.

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

Fortsetzung von vorheriger Seite ...

Da die Knoten zur Auswertung des bedingten Ausdrucks „if (not(exists(\$x/price))) then \$x else ()“ nun vorliegen, wertet der *Anfrageevaluator* diesen aus. Es erfolgt keine Ausgabe (else-Fall), da der if-Fall „false“ ergibt. Folgend wird nun die nach dem bedingten Ausdruck stehende Sequenz von SignOff-Anweisungen vom *Anfrageevaluator* bearbeitet.

- `signOff($x, r1)` führt dazu, dass der *Speichermanager* die Rolle r_1 von der momentanen Bindung der Variablen x , d.h. vom gespeicherten book-Knoten, entfernt.
- `signOff($x/price[1], r2)` führt dazu, dass der *Speichermanager* die Rolle r_2 des gespeicherten price-Knotens entfernt.
- `signOff($x/dos::node(), r3)` führt dazu, dass der *Speichermanager* die Rolle r_3 aller Nachkommen des gespeicherten book-Knotens und von diesem selbst entfernt.

Die gespeicherten Knoten *year*, *author* und Nachkommen (*last* und *first*), *publisher* und *price* haben alle ihre Rollen verloren. Da sie irrelevant – wie Definition 3.14 vorschreibt – sind, also keine Nachkommen, die mit einer Rolle versehen sind, besitzen, werden sie vom *Speichermanager* samt ihrer zugehörigen Textknoten aus dem Speicher entfernt.

Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

SCHRITT 13	
<i>Eingabedatenstrom:</i>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	<pre> bib{r0} book{r4} title{r5} "TCP/IP Illustrated" </pre>
<p>Alle sich nun noch im Speicher befindenden Knoten besitzen eine Rolle, die sie als relevant für die Auswertung der noch ausstehenden <i>for_{\$b}</i>- und <i>return</i>-Klausel kenntlich macht. Zuvor muss der <i>Anfrageevaluator</i> allerdings auf neue Eingabe warten, da die Bearbeitung der <i>for_{\$x}</i>-Klausel noch nicht beendet ist.</p>	
SCHRITT 14	
<i>Eingabedatenstrom:</i> </book>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	<pre> bib{r0} book{r4} title{r5} "TCP/IP Illustrated" </pre>
<p>Der <i>Dokumentprojektor</i> liest den schließenden Tag </book>.</p>	
<p><i>Fortsetzung auf der nächsten Seite ...</i></p>	

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

SCHRITT 15	
<i>Eingabedatenstrom:</i>	<code></bib></code>
<i>Ausgabedatenstrom:</i>	<code><title>TCP/IP Illustrated</title></code>
<i>Speicherinhalt:</i>	<pre style="text-align: center;"> bib{r0} book{r4} title{r5} "TCP/IP Illustrated" </pre>
<p>Der <i>Dokumentprojektor</i> liest den schließenden Tag <code></bib></code>, der den Speicher vervollständigt bzw. das Ende des Datenstroms ist. Da die Variable $\\$x$ kein zweites Mal an einen (gespeicherten) Knoten gebunden werden kann, wird mit der Evaluation der <i>for_{\$b}</i>-Klausel seitens des <i>Anfrageevaluators</i> fortgefahren. Von ihm wird die Variable $\\$b$ an den book-Knoten, der sich immer noch im Speicher befindet, gebunden und der Körper der return-Klausel bzw. der nach ihr stehende Ausdruck führt dazu, dass der title-Knoten, der sich ebenfalls noch im Speicher befindet, ausgegeben wird. Folgend wird nun in den nächsten Schritten die im Körper der return- der <i>for_{\$b}</i>-Klausel enthaltene Sequenz von SignOff-Anweisungen:</p> <ul style="list-style-type: none"> • <code>signOff($\\$b$, r_4)</code> • <code>signOff($\\$b$/title/dos::node(), r_5),</code> <p>vom <i>Anfrageevaluator</i> bearbeitet.</p>	
<i>Fortsetzung auf der nächsten Seite ...</i>	

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

Fortsetzung von vorheriger Seite ...

SCHRITT 16	
<i>Eingabedatenstrom:</i>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	$ \begin{array}{c} \text{bib}\{r_0\} \\ \\ \text{book}\{\#4\} \\ \\ \text{title}\{r_5\} \\ \\ \text{"TCP/IP Illustrated"} \end{array} $
<p>Die erste SignOff-Anweisung, $\text{signOff}(\\$b, r_4)$, führt dazu, dass der <i>Speichermanager</i> die einzige Rolle r_4 von der momentanen Bindung der Variablen $\\$b$, d.h. vom gespeicherten book-Knoten, entfernt. Da dieser zwar keine weiteren Rollen, aber noch Nachkommen, die mit einer Rolle versehen sind, besitzt, wird er – da er nach Definition 3.14 noch relevant ist – nicht aus dem Speicher entfernt.</p>	
SCHRITT 17	
<i>Eingabedatenstrom:</i>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	$ \begin{array}{c} \text{bib}\{r_0\} \\ \\ \del{\text{book}}\{\#4\} \\ \\ \del{\text{title}}\{\#6\} \\ \\ \text{"TCP/IP Illustrated"} \end{array} $

Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

Fortsetzung von vorheriger Seite ...

Die zweite SignOff-Anweisung, $\text{signOff}(\$b/\text{title}/\text{dos}::\text{node}(), r_5)$, führt dazu, dass der *Speichermanager* die einzige Rolle r_5 des gespeicherten title-Knotens entfernt. Da dieser keine weiteren Rollen und keine Nachkommen, die noch mit einer Rolle versehen sind, besitzt, wird er – da er nach Definition 3.14 irrelevant ist – vom *Speichermanager* zusammen mit seinem zugehörigen Textknoten aus dem Speicher entfernt. Dies hat zur Folge, dass auch der book-Knoten, da dieser seine Rolle schon früher verloren hat (s. Schritt 16) und nun ebenfalls keine Nachkommen, die mit einer Rolle versehen sind, besitzt, vom *Speichermanager* entfernt wird.

SCHRITT 18

Eingabedatenstrom:

Ausgabedatenstrom:

Speicherinhalt:

bib^{r_0}

Da die Variable $\$b$ kein zweites Mal an einen (gespeicherten) Knoten gebunden werden kann, wird die in der Anfrage enthaltene SignOff-Anweisung:

- $\text{signOff}(\$bib, r_0)$

vom *Anfrageevaluator* bearbeitet.

Fortsetzung auf der nächsten Seite ...

Tabelle C.1: Aktive Speicherbereinigung für Beispiel 2.1 und Beispiel 3.17
(Fortsetzung)

Fortsetzung von vorheriger Seite ...	
SCHRITT 19	
<i>Eingabedatenstrom:</i>	
<i>Ausgabedatenstrom:</i>	
<i>Speicherinhalt:</i>	bib {#b}
<p>Diese SignOff-Anweisung, <code>signOff(\$bib, r₀)</code>, führt dazu, dass der <i>Speichermanager</i> die einzige Rolle r_0 von der momentanen Bindung der Variablen <code>\$bib</code>, d.h. vom gespeicherten bib-Knoten, entfernt. Da dieser keine weiteren Rollen mehr besitzt, er der letzte Knoten im Speicher ist und somit keine Nachkommen besitzt, die noch mit einer Rolle versehen sind, wird auch er – da er nach Definition 3.14 irrelevant ist – vom <i>Speichermanager</i> aus dem Speicher entfernt.</p>	
SCHRITT 20	
<i>Eingabedatenstrom:</i>	
<i>Ausgabedatenstrom:</i>	</r>
<i>Speicherinhalt:</i>	\emptyset
<p>Da die Variable <code>\$bib</code> kein zweites Mal an einen (gespeicherten) Knoten gebunden werden kann, ist die Auswertung der Anfrage beendet. Der Speicher ist, bevor zuletzt der schließende Tag <code></r></code> ausgegeben wird, leer. Somit ist die Auswertung der Anfrage aus Beispiel 3.17 auf das in Beispiel 2.1 enthaltene Dokument, da alle Rollen, die entfernt wurden, definiert waren und alle Knoten nach Beendigung der Auswertung aus dem Speicher entfernt sind, nach Definition 3.8 <i>sicher</i>.</p>	

Anhang D

GCX Erweiterung: Pfade mit mehreren Schritten

D.1 XML-Dokument

```
<bib>
  <book>
    <year>1994</year>
    <title>TCP/IP Illustrated</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
    <toc>
      <chapter>
        <title>1. Introduction</title>
        <section><title>1.1 Introduction</title></section>
        <section><title>1.2 Layering</title></section>
        <section><title>1.3 TCP/IP Layering</title></section>
        <section><title>1.4 Internet Addresses</title></section>
      </chapter>
    </toc>
  </book>
  <book>
    <year>1992</year>
    <title>Advanced Programming In The Unix Environment</title>
    <author><last>Stevens</last><first>W.</first></author>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
    <toc>
      <chapter>
        <title>2. Unix Standardization And Implementations</title>
        <section><title>2.1 Introduction</title></section>
        <section>
          <title>2.2 Unix Standardization</title>
          <section><title>2.2.1 ANSI C</title></section>
          <section><title>2.2.2 IEEE POSIX</title></section>
          <section><title>2.2.3 X/Open XPG3</title></section>
          <section><title>2.2.4 FIPS</title></section>
        </section>
      </chapter>
    </toc>
  </book>
</bib>
```

```

        </section>
        <section>
            <title>2.3 Unix Implementations</title>
            <section><title>2.3.1 System V Release 4</title></section>
            <section><title>2.3.2 4.3+BSD</title></section>
        </section>
    </chapter>
</toc>
</book>
<book>
    <year>2000</year>
    <title>Data On The Web</title>
    <author><last>Abiteboul</last><first>Serge</first></author>
    <author><last>Buneman</last><first>Peter</first></author>
    <author><last>Suciu</last><first>Dan</first></author>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
    <toc>
        <chapter>
            <title>3. XML</title>
            <section>
                <title>3.1 Basic Syntax</title>
                <section><title>3.1.1 XML Elements</title></section>
                <section><title>3.1.2 XML Attributes</title></section>
                <section><title>3.1.3 Well-Formed XML Documents</title></section>
            </section>
            <section>
                <title>3.2 XML And Semistructured Data</title>
                <section><title>3.2.1 XML Graph Model</title></section>
                <section><title>3.2.2 XML References</title></section>
                <section><title>3.2.3 Order</title></section>
                <section><title>3.2.4 Mixing Elements And Text</title></section>
            </section>
            <section>
                <title>3.3 Document Type Definitions</title>
                <section><title>3.3.1 A Simple DTD</title></section>
                <section><title>3.3.2 DTDs As Grammars</title></section>
                <section><title>3.3.3 DTDs As Schemas</title></section>
                <section><title>3.3.4 Declaring Attributes In DTDs</title></section>
            </section>
        </chapter>
    </toc>
</book>
</bib>

```

Quelltext D.1: Erweitertes XML-Dokument für eine Bibliothek

Anhang E

GCX Erweiterung: Aggregatfunktionen

E.1 XML-Dokument

```
<borrower>
  <person>
    <gender>female</gender>
    <name>Xinan Hiltgen</name>
    <address>
      <street>83 Littlestone St</street>
      <city>Bologna</city>
      <country>United States</country>
      <province>North Carolina</province>
      <zipcode>18</zipcode>
    </address>
    <emailaddress>mailto:Hiltgen@uiuc.edu</emailaddress>
    <phone>+64 (612) 64745468</phone>
    <age>20</age>
    <creditcard>1992 8716 1756 5856</creditcard>
    <account>
      <amount>9876.00</amount>
      <valid><from>2000-07-05</from><to>2000-12-05</to></valid>
    </account>
  </person>
  <person>
    <gender>male</gender>
    <name>Wonchan Pacholski</name>
    <address>
      <street>10 Darroux St</street>
      <city>Merida</city>
      <country>Saint Kitts</country>
      <province>Rachid</province>
      <zipcode>32</zipcode>
    </address>
    <emailaddress>mailto:Pacholski@baylor.edu</emailaddress>
    <phone>+0 (301) 3201232</phone>
    <age>29</age>
    <creditcard>6014 9922 1367 9230</creditcard>
    <account>
      <amount>31437.24</amount>
```

```

    <valid><from>2001-05-28</from><to>2001-10-28</to></valid>
  </account>
</person>
<person>
  <gender>female</gender>
  <name>Changming Desai</name>
  <address>
    <street>39 Hatonen St</street>
    <city>Orlando</city>
    <country>United States</country>
    <province>Ohio</province>
    <zipcode>14</zipcode>
  </address>
  <emailaddress>mailto:Desai@poznan.pl</emailaddress>
  <phone>+0 (968) 64035662</phone>
  <age>63</age>
  <creditcard>7139 3533 7124 2305</creditcard>
  <account>
    <amount>44518.90</amount>
    <valid><from>1998-04-06</from><to>1998-09-06</to></valid>
  </account>
</person>
<person>
  <gender>male</gender>
  <name>Tonny Matzat</name>
  <address>
    <street>4 Scallan St</street>
    <city>Boise</city>
    <country>United States</country>
    <province>Massachusetts</province>
    <zipcode>33</zipcode>
  </address>
  <emailaddress>mailto:Matzat@washington.edu</emailaddress>
  <phone>+0 (665) 80556017</phone>
  <age>18</age>
  <creditcard>6074 7747 1572 5867</creditcard>
  <account>
    <amount>12106.89</amount>
    <valid><from>1999-12-16</from><to>2000-05-16</to></valid>
  </account>
</person>
</borrower>

```

Quelltext E.1: XML-Dokument für eine Personenliste von Entleiher

Anhang F

GCX Optimierungen

F.1 XML-Dokument

```
<loan>
  <books>
    <book>
      <year>1994</year>
      <title>TCP/IP Illustrated</title>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
      <borrower>
        <person>
          <gender>female</gender>
          <name>Xinan Hiltgen</name>
          <emailaddress>mailto:Hiltgen@uiuc.edu</emailaddress>
          <creditcard>1992 8716 1756 5856</creditcard>
        </person>
        <date>
          <from>2000-08-04</from>
          <to>2000-08-25</to>
        </date>
        <fee>197.85</fee>
      </borrower>
    </book>
    <book>
      <year>1992</year>
      <title>Advanced Programming In The Unix Environment</title>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
      <borrower>
        <person>
          <gender>male</gender>
          <name>Wonchan Pacholski</name>
          <emailaddress>mailto:Pacholski@baylor.edu</emailaddress>
          <creditcard>6014 9922 1367 9230</creditcard>
        </person>
        <date>
          <from>2001-06-27</from>
          <to>2001-08-01</to>
        </date>
      </borrower>
    </book>
  </books>
</loan>
```

```

        </date>
        <fee>329.75</fee>
    </borrower>
</book>
<book>
    <year>2000</year>
    <title>Data On The Web</title>
    <publisher>Morgan Kaufmann Publishers</publisher>
    <price>39.95</price>
    <borrower>
        <person>
            <gender>female</gender>
            <name>Changming Desai</name>
            <emailaddress>mailto:Desai@poznan.pl</emailaddress>
            <creditcard>7139 3533 7124 2305</creditcard>
        </person>
        <date>
            <from>1998-05-05</from>
            <to>1998-05-19</to>
        </date>
        <fee>79.9</fee>
    </borrower>
</book>
<book>
    <year>1999</year>
    <title>The Economics Of Technology And Content For Digital TV</title>
    <publisher>Kluwer Academic Publishers</publisher>
    <price>129.95</price>
    <borrower>
        <person>
            <gender>male</gender>
            <name>Tonny Matzat</name>
            <emailaddress>mailto:Matzat@washington.edu</emailaddress>
            <creditcard>6074 7747 1572 5867</creditcard>
        </person>
        <date>
            <from>2000-01-15</from>
            <to>2000-02-12</to>
        </date>
        <fee>519.8</fee>
    </borrower>
</book>
</books>
</loan>

```

Quelltext F.1: XML-Dokument von entliehenen Büchern und deren Entleiher

Anhang G

Experimentelle Resultate

G.1 XMark Testanfragen

G.1.1 Pfade mit einem Schritt

G.1.1.1 XMark-Anfrage Q01

```
<query01> {  
  for $site in /site return  
    for $people in $site/people return  
      for $person in $people/person  
        where($person/person_id = "person0") return  
          for $name in $person/name return  
            $name/text()  
}  
</query01>
```

Quelltext G.1: XMark-Anfrage Q01 mit Pfaden mit einem Schritt

G.1.1.2 XMark-Anfrage Q02

```
<query02> {  
  for $site in /site return  
    for $open_auctions in $site/open_auctions return  
      for $open_auction in $open_auctions/open_auction return  
        <increase> {  
          for $bidder in $open_auction/bidder return  
            for $increase in $bidder/increase return  
              <bid> {$increase/text()} </bid>  
        } </increase>  
}  
</query02>
```

Quelltext G.2: XMark-Anfrage Q02 mit Pfaden mit einem Schritt

G.1.1.3 XMark-Anfrage Q08

```

<query08> {
  for $site in /site return
    for $people in $site/people return
      for $person in $people/person return
        (<person> {for $name in $person/name return $name/text()} </person>,
         <items_bought> {
           for $site in /site return
             for $closed_auctions in $site/closed_auctions return
               for $closed_auction in $closed_auctions/closed_auction return
                 for $buyer in $closed_auction/buyer
                   where ($buyer/buyer_person = $person/person_id) return
                     $closed_auction
           } </items_bought>)
} </query08>

```

Quelltext G.3: XMark-Anfrage Q08 mit Pfaden mit einem Schritt

G.1.1.4 XMark-Anfrage Q09

```

<query09> {
  for $site in /site return
    for $people in $site/people return
      for $person in $people/person return
        (<person>
         <name> {for $name in $person/name return $name/text()} </name>
         {
           for $site in /site return
             for $closed_auctions in $site/closed_auctions return
               for $closed_auction in $closed_auctions/closed_auction return
                 for $buyer in $closed_auction/buyer
                   where ($person/person_id = $buyer/buyer_person) return
                     for $site in /site return
                       for $regions in $site/regions return
                         for $europe in $regions/europe return
                           for $item in $europe/item return
                             for $itemref in $closed_auction/itemref
                               where ($itemref/itemref_item = $item/item_id)
                                 return
                                   <item> {
                                     for $name in $item/name return $name/text()
                                   } </item>
                             }
         } </person>)
} </query09>

```

Quelltext G.4: XMark-Anfrage Q09 mit Pfaden mit einem Schritt

G.1.1.5 XMark-Anfrage Q11

```

<query11> {
  for $site in /site return
    for $people in $site/people return
      for $person in $people/person return
        (<items>
          <name> {for $name in $person/name return $name/text()} </name>
          <initial> {
            for $site in /site return
              for $open_auctions in $site/open_auctions return
                for $open_auction in $open_auctions/open_auction return
                  for $initial in $open_auction/initial return
                    for $profile in $person/profile
                      where ($profile/profile_income > "5000") return
                        $initial
          } </initial>
        ) </items>
} </query11>

```

Quelltext G.5: XMark-Anfrage Q11 mit Pfaden mit einem Schritt

G.1.1.6 XMark-Anfrage Q12

```

<query12> {
  for $site in /site return
    for $people in $site/people return
      for $person in $people/person return
        for $profile in $person/profile
          where ($profile/profile_income > "50000") return
            (<items>
              <person> {
                for $profile in $person/profile return $profile/profile_income
              } </person>
              <initial> {
                for $site in /site return
                  for $open_auctions in $site/open_auctions return
                    for $open_auction in $open_auctions/open_auction return
                      for $initial in $open_auction/initial return
                        for $profile in $person/profile
                          where ($profile/profile_income > "5000") return
                            $initial
              } </initial>
            ) </items>
} </query12>

```

Quelltext G.6: XMark-Anfrage Q12 mit Pfaden mit einem Schritt

G.1.1.7 XMark-Anfrage Q13

```
<query13> {
  for $site in /site return
    for $regions in $site/regions return
      for $australia in $regions/australia return
        for $item in $australia/item return
          <item>
            <name> {for $name in $item/name return $name/text()} </name>
            <description> {$item/description} </description>
          </item>
} </query13>
```

Quelltext G.7: XMark-Anfrage Q13 mit Pfaden mit einem Schritt

G.1.1.8 XMark-Anfrage Q14

```
<query14> {
  for $site in /site return
    for $item in $site//item
      where ($item/description <= "foo") return
        for $name in $item/name return $name/text()
} </query14>
```

Quelltext G.8: XMark-Anfrage Q14 mit Pfaden mit einem Schritt

G.1.1.9 XMark-Anfrage Q15

```
<query15> {
  for $site in /site return
    for $closed_auctions in $site/closed_auctions return
      for $closed_auction in $closed_auctions/closed_auction return
        for $annotation in $closed_auction/annotation return
          for $description in $annotation/description return
            for $parlist in $description/parlist return
              for $listitem in $parlist/listitem return
                for $parlist in $listitem/parlist return
                  for $listitem in $parlist/listitem return
                    for $text in $listitem/text return
                      for $emph in $text/emph return
                        for $keyword in $emph/keyword return
                          for $text in $keyword/text() return
                            <text> {$text} </text>
} </query15>
```

Quelltext G.9: XMark-Anfrage Q15 mit Pfaden mit einem Schritt

G.1.1.10 XMark-Anfrage Q18

```
<query18> {  
  for $site in /site return  
    for $open_auctions in $site/open_auctions return  
      for $open_auction in $open_auctions/open_auction return  
        $open_auction/reserve  
}  
</query18>
```

Quelltext G.10: XMark-Anfrage Q18 mit Pfaden mit einem Schritt

G.1.1.11 XMark-Anfrage Q19

```
<query19> {  
  for $site in /site return  
    for $regions in $site/regions return  
      for $item in $regions//item return  
        <item>  
          <name> {for $name in $item/name return $name/text()} </name>  
          {  
            for $location in $item/location return  
              $location/text()  
          }  
        </item>  
}  
</query19>
```

Quelltext G.11: XMark-Anfrage Q19 mit Pfaden mit einem Schritt

G.1.2 Pfade mit mehreren Schritten

G.1.2.1 XMark-Anfrage Q01

```
<query01> {
  for $person in /site/people/person
    where($person/person_id = "person0") return
      $person/name/text()
} </query01>
```

Quelltext G.12: XMark-Anfrage Q01 mit Pfaden mit mehreren Schritten

G.1.2.2 XMark-Anfrage Q02

```
<query02> {
  for $open_auction in /site/open_auctions/open_auction return
    <increase> {
      for $increase in $open_auction/bidder/increase return
        <bid> {$increase/text()} </bid>
    } </increase>
} </query02>
```

Quelltext G.13: XMark-Anfrage Q02 mit Pfaden mit mehreren Schritten

G.1.2.3 XMark-Anfrage Q03

```
<query03> {
  for $open_auction in /site/open_auctions/open_auction
    where ($open_auction/bidder/increase/text() <= $open_auction/bidder/
      increase/text()) return
      <increase> {
        for $increase in $open_auction/bidder/increase return
          <bid> {$increase/text()} </bid>
      } </increase>
} </query03>
```

Quelltext G.14: XMark-Anfrage Q03 mit Pfaden mit mehreren Schritten

G.1.2.4 XMark-Anfrage Q05

```
<query05> {  
  fn:count(/site/closed_auctions/closed_auction/price)  
} </query05>
```

Quelltext G.15: XMark-Anfrage Q05 mit Pfaden mit mehreren Schritten

G.1.2.5 XMark-Anfrage Q06

```
<query06> {  
  for $regions in //site/regions return  
    fn:count($regions//item)  
} </query06>
```

Quelltext G.16: XMark-Anfrage Q06 mit Pfaden mit mehreren Schritten

G.1.2.6 XMark-Anfrage Q07

```
<query07> {  
  for $site in /site return  
  (  
    <count_description> {fn:count($site//description)} </count_description>,  
    <count_annotation> {fn:count($site//annotation)} </count_annotation>,  
    <count_emailaddress> {fn:count($site//emailaddress)} </count_emailaddress>  
  )  
} </query07>
```

Quelltext G.17: XMark-Anfrage Q07 mit Pfaden mit mehreren Schritten

G.1.2.7 XMark-Anfrage Q08

```
<query08> {  
  for $person in /site/people/person return  
    (<person> {$person/name/text()} </person>,  
    <items_bought> {  
      for $closed_auction in /site/closed_auctions/closed_auction  
        where ($closed_auction/buyer/buyer_person = $person/person_id) return  
          $closed_auction  
    } </items_bought>)  
} </query08>
```

Quelltext G.18: XMark-Anfrage Q08 mit Pfaden mit mehreren Schritten

G.1.2.8 XMark-Anfrage Q09

```

<query09> {
  for $person in /site/people/person return
    (<person><name> {$person/name/text()} </name>
    {
      for $closed_auction in /site/closed_auctions/closed_auction
        where ($person/person_id = $closed_auction/buyer/buyer_person) return
          for $item in /site/regions/europe/item
            where ($closed_auction/itemref/itemref_item = $item/item_id)
              return <item> {$item/name/text()} </item>
    } </person>)
} </query09>

```

Quelltext G.19: XMark-Anfrage Q09 mit Pfaden mit mehreren Schritten

G.1.2.9 XMark-Anfrage Q10

```

<query10> {
  for $interest_category in /site/people/person/profile/interest/
    interest_category return
    (<categorie><id> {$interest_category} </id>
    {
      for $person in /site/people/person
        where ($person/profile/interest/interest_category
          = $interest_category) return
          <personne>
            <statistiques>
              <sexe> {$person/profile/gender/text()} </sexe>
              <age> {$person/profile/age/text()} </age>
              <education> {$person/profile/education/text()} </education>
              <revenu> {$person/profile/profile_income} </revenu>
            </statistiques>
            <coordonnees>
              <nom> {$person/name/text()} </nom>
              <rue> {$person/address/street/text()} </rue>
              <ville> {$person/address/city/text()} </ville>
              <pays> {$person/address/country/text()} </pays>
              <reseau>
                <courrier> {$person/emailaddress/text()} </courrier>
                <pagePerso> {$person/homepage/text()} </pagePerso>
              </reseau>
            </coordonnees>
            <cartePaiement> {$person/creditcard/text()} </cartePaiement>
          </personne>
    } </categorie>)
} </query10>

```

Quelltext G.20: XMark-Anfrage Q10 mit Pfaden mit mehreren Schritten

G.1.2.10 XMark-Anfrage Q11

```
<query11> {
  for $person in /site/people/person return
    (<items>
      <name> {$person/name/text()} </name>
      <initial> {
        for $initial in /site/open_auctions/open_auction/initial
          where ($person/profile/profile_income > "5000") return
            $initial
        } </initial>
      </items>)
} </query11>
```

Quelltext G.21: XMark-Anfrage Q11 mit Pfaden mit mehreren Schritten

G.1.2.11 XMark-Anfrage Q12

```
<query12> {
  for $person in /site/people/person
    where ($person/profile/profile_income > "50000") return
    (<items>
      <person> {$person/profile/profile_income} </person>
      <initial> {
        for $initial in /site/open_auctions/open_auction/initial
          where ($person/profile/profile_income > "5000") return
            $initial
        } </initial>
      </items>)
} </query12>
```

Quelltext G.22: XMark-Anfrage Q12 mit Pfaden mit mehreren Schritten

G.1.2.12 XMark-Anfrage Q13

```
<query13> {
  for $item in /site/regions/australia/item return
    <item>
      <name> {$item/name/text()} </name>
      <description> {$item/description} </description>
    </item>
} </query13>
```

Quelltext G.23: XMark-Anfrage Q13 mit Pfaden mit mehreren Schritten

G.1.2.13 XMark-Anfrage Q14

```
<query14> {
  for $item in /site//item
    where ($item/description <= "foo") return
      $item/name/text()
} </query14>
```

Quelltext G.24: XMark-Anfrage Q14 mit Pfaden mit mehreren Schritten

G.1.2.14 XMark-Anfrage Q15

```
<query15> {
  for $text in /site/closed_auctions/closed_auction/annotation/description/
    parlist/listitem/parlist/listitem/text/emph/keyword/text() return
    <text> {$text} </text>
} </query15>
```

Quelltext G.25: XMark-Anfrage Q15 mit Pfaden mit mehreren Schritten

G.1.2.15 XMark-Anfrage Q16

```
<query16> {
  for $closed_auction in /site/closed_auctions/closed_auction
    where ($closed_auction/annotation/description/parlist/listitem/parlist/
      listitem/text/emph/keyword/text() != "") return
    <person>
      <id> {$closed_auction/seller/seller_person} </id>
    </person>
} </query16>
```

Quelltext G.26: XMark-Anfrage Q16 mit Pfaden mit mehreren Schritten

G.1.2.16 XMark-Anfrage Q17

```
<query17> {
  for $person in /site/people/person
    where (fn:not(fn:exists($person/homepage/text()))) return
    <person>
      <name> {$person/name/text()} </name>
    </person>
} </query17>
```

Quelltext G.27: XMark-Anfrage Q17 mit Pfaden mit mehreren Schritten

G.1.2.17 XMark-Anfrage Q18

```
<query18> {  
  for $open_auction in /site/open_auctions/open_auction return  
    $open_auction/reserve  
} </query18>
```

Quelltext G.28: XMark-Anfrage Q18 mit Pfaden mit mehreren Schritten

G.1.2.18 XMark-Anfrage Q19

```
<query19> {  
  for $item in /site/regions//item return  
    <item><name> {for $name in $item/name return $name/text()} </name>  
    {$item/location/text()}  
  </item>  
} </query19>
```

Quelltext G.29: XMark-Anfrage Q19 mit Pfaden mit mehreren Schritten

G.1.2.19 XMark-Anfrage Q20

```
<query20>  
  <count_profiles> {fn:count(/site/people/person/profile)} </count_profiles>  
  <income_info>  
    <sum_income> {fn:sum(/site/people/person/profile/profile_income)} </sum_income>  
    <avg_income> {fn:avg(/site/people/person/profile/profile_income)} </avg_income>  
    <min_income> {fn:min(/site/people/person/profile/profile_income)} </min_income>  
    <max_income> {fn:max(/site/people/person/profile/profile_income)} </max_income>  
  </income_info>  
  <age_info>  
    <sum_age> {fn:sum(/site/people/person/profile/age)} </sum_age>  
    <avg_age> {fn:avg(/site/people/person/profile/age)} </avg_age>  
    <min_age> {fn:min(/site/people/person/profile/age)} </min_age>  
    <max_age> {fn:max(/site/people/person/profile/age)} </max_age>  
  </age_info>  
</query20>
```

Quelltext G.30: XMark-Anfrage Q20 mit Pfaden mit mehreren Schritten