INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen

Oettingenstraße 67 D–80538 München

Use Cases für Xcerpt: Eine positionelle Anfrage- und Transformationssprache für das Web

Sebastian Kraus

Diplomarbeit

Beginn der Arbeit:	01.10.2003
Abgabe der Arbeit:	31.03.2004
Betreuer:	Prof. Dr. François Bry
	Dipl.-Inform. Sebastian Schaffert

Erklärung

Hiermit versichere ich, dass ich diese Diplomarbeit selbständig verfraßt habe. Ich habe dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwandt.

München, den 31. März 2004 Sebastian Kraus

Zusammenfassung

In dieser Arbeit werden Anwendungsfälle für die logische Anfragesprache Xcerpt vorgestellt und implementiert. Zweck dieser Arbeit ist hierbei, die Sprache Xcerpt, welche sich noch in einem Entwicklungszustand befindet, bezüglich ihrer Ausdrucksfähigkeit und Mächtigkeit zu untersuchen und gegebenenfalls notwendige und noch fehlende Konstrukte und Funktionalitäten zu identifizieren.

Hierbei werden zum einem die W3C Anwendungsfälle für XQuery in Xcerpt implementiert, da diese wegen ihrer Vielzahl von unterschiedlichen Szenarien und somit unterschiedlichen Anfragen als Maßstab für eine Anfragesprache herangezogen werden können. Zum anderen werden drei kleiner Anwendungsszenarien vorgestellt, die spezielle Fähigkeiten von Xcerpt hervorheben sollen.

In dem ersten dieser Szenarien wird der internen Referenzmechanismus von Xcerpt vorgestellt. Da Xcerpt als Webanfragsprache konzipiert wurde, wird in der zweiten Anwendung ein Webcrawler vorgestellt. Die dritte Anwendung beinhaltet Xcerpt-Regel, wie sie ein Mediatorsystem verwenden könnte, um die Transformationsmöglichkeiten der Sprache hervorzuheben.

Danksagung

Für die sehr gute Betreuung und die angenehme Atmosphäre bei meiner Diplomarbeit möchte ich mich bei meinen Betreuern Prof. Dr. François Bry und Sebastian Schaffert und dem gesamten Team der Lehr- und Forschungseinheit für Programmier- und Modellierungssprachen bedanken.

Im Besonderen möchte ich mich bei Paula Pàtrânjan, Julia Palchaninava und Ekaterina Savenkova bedanken, die mich zu jeder Zeit dieser Arbeit moralisch unterstützt und ermutigt haben.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Seit dem Aufkommen von XML als Sprache zur Repräsentation semistrukturierter Daten wächst gleichzeitig der Bedarf nach einer Sprache, die (semi-)strukturierte Dokumente befragen kann. Mittlerweile existieren mehrere Anfragesprachen für semistrukturierte Daten, u.a. XSLT [JC99] und XQuery [SBDCMFF+03]. Diese unterscheiden sich mehr oder weniger stark durch ihre Ausdrucksmächtigkeit, durch unterschiedliche Funktionalitäten und durch Konzepte.

Häufig haben sich aus Anforderungen der Industrie diese Sprachen entwickelt und sind deswegen besonders auf bestimmte Anwendungen zugeschnitten. Aber nicht jede Sprache ist für jede Art von Anwendung geeignet. Eine Initiative, die vom W3C[1] ausgeht, hat es sich zur Aufgabe gemacht die unterschiedlichen Anforderungen an eine Anfragesprache und im Speziellen an eine XML-Anfragesprache hervorheben.

Eine mittlerweile etablierte Anfragesprache ist XQuery. Sie ist in intensiver Zusammenarbeit von Forschung und Industrie entstanden und speziell für Anfragen auf XML Dokumente entwickelt worden. Jede Anfrage wird als ein Ausdruck angesehen, der berechnet wird. Ausdrücke können mit anderen Ausdrücken kombiniert werden um neue Ausdrücke zu erzeugen. XQuery wird daher als eine funktionale Sprache angesehen [DCDDMF+03]. XQuery ist eine pfadorientierte Sprache und nutzt zu Pfadselektion XPath [JCSD99]. Zum Konstruieren von neuen Daten werden Patterns verwendet, die angeben, wie und in welcher Form die selektierten Daten ausgegeben werden sollen. In XQuery sind Selektion und Konstruktion miteinander in den Ausdrücken verflochten [FBSS02a]. Für überschaubare Anfragen ist diese Verflechtung zusammen mit der pfadorientierte Selektion eine überschaubare und natürliche Art der Programmierung. Bei komplexeren Anfragen in Verbindung mit vielen Variablen in den Ausdrücken, werden diese jedoch schnell unübersichtlich.

Eine Anfragesprache, die zur Selektion von Daten nicht einen pfadorientierte Ansatz verfolgt, ist UnQL [BDS95]. Dort werden Patterns verwendet, die mit einer Art Pattern-Matching-Algorithmus auf Datenbankinhalte "gelegt" werden. Ein Pattern kann man als eine Schablone auf eine Datenbank betrachten, der um Variablen zur Selektion von Daten angereichert ist.

[1] http://www.w3c.org

Bei der Anfrage werden die Variablen eines Patterns an die entsprechenden Daten gebunden. Zur Konstruktion des gewünschten Ergebnisses wird wiederum ein Pattern verwendet. Im Gegensatz zu XQuery gibt es in UnQL eine strikte Trennung zwischen Selektion und Konstruktion (siehe [SBFBSS03]). Diese Trennung von Selektion und Konstruktion ist vergleichbar mit Regeln aus der logischen Programmiersprache Prolog[2] und mit Datenbanksichten, wie sie die Anfragesprache SQL[3] besitzt. So ist der Anfrageteil vergleichbar mit einem Prolog Ziel, während der Konstruktionsteil dem Kopf einer Prolog Regel entspricht. Aus der Überzeugung heraus, dass eine Anfragesprache deklarativer ist, wenn patternbasierten Anfragen und Paradigmen aus der Logikprogrammierung vereint werden, ist die Anfragesprache Xcerpt[4] entstanden, auf der diese Arbeit aufbaut (siehe hierzu [FBSS02a]).

1.2 Die Idee von Xcerpt, eine Sprache zur Anfrage des Web

Die größte existierende Datenbank und Wissensplattform ist wohl das World Wide Web, welches aus der Datenbanksicht ein entscheidendes Manko hat; es gibt weder ein genormtes Datenformat noch unterliegen die verteilten Daten einem einheitlichen Schema. Aus dieser Erkenntnis heraus ist auch XML entstanden, um wenigstens einen Standard zur Strukturierung von Daten zu schaffen, auch wenn diese aus unvermeidlichen Gründen auf unterschiedlichen Schemata basieren. Mit XML wird eine Syntax für Baum- und Graphstrukturierte Daten festgelegt die zwei wichtige Eigenschaften erfüllen müssen: Die Daten sollen semistrukturiert und wohlgeformt sein.

Mit herkömmlichen Datenbankanfragesprachen ist die Extraktion von Wissen aus dem Web nicht zu meistern, da man zu diesem Zweck zumindest Kenntnis über die unterschiedlichen Schemata besitzen muss. Aus diesem Grund haben sich im Bereich der Informatik Forschungsbereiche entwickelt, die es sich zur Aufgabe gemacht haben, diese Daten zu extrahieren, zu kombinieren und daraus neues Wissen zu generieren, was durch das entstehen einer “Semantic Web”-Gemeinde hervorgehoben wird. Diese Art von Wissensextraktion und Wissensgenerierung nennt man auch “automated reasoning”.

Dazu benötigt man allerdings eine geeignete Anfragesprache, die mit unbekannten, unvollständigen und unterschiedlichen Daten umgehen kann. Aus der Sicht eines Programmierers bedeutet dies, dass er so wenig wie möglich angeben muss, wie er zu seinem Wissen kommt und stattdessen sich so weit wie möglich auf die Spezifikation der Information, die er erhalten will, beschränken kann. Dies erfordert eine sehr deklarative Anfragesprache.

Wie in der Einleitung vorgestellt, basiert XQuery, die wohl am weitesten entwickelte Anfragesprache für XML, auf Pfadangaben zur Selektion von Daten. Dies bedeutet, dass man den Weg, um zu den Daten zu gelangen vorgibt. Man navigiert sich also durch den Baum oder Graph, indem die Daten strukturiert sind. Diese Art der expliziten Navigation kann man durchaus als Einschränkung der Deklarativität sehen (siehe hierzu [FBSS02c]). UnQL und Xcerpt hingegen erlauben es durch Muster das Aussehen der gewünschten Daten zu spezifizieren. Wie man die Daten erhält, bleibt dem Auswertungsalgorithmus der Sprache überlassen. Wegen der Ähnlichkeit mit Regeln aus der logischen Programmierung erhält man

[2] http://www.swi-prolog.org
[3] http://www.w3schools.com/sql
[4] http://www.xcerpt.org

alle möglichen Variablenbindungen, die diesem Muster entsprechen.

In der Anfragesprache Xcerpt werden drei Ideen verwirklicht (siehe [FBSS02b]):

Xcerpt basiert auf Pattern: Die Idee des Pattern Ansatzes ist, dass der Benutzer ein exemplarisches, eventuell unvollständiges Beispieldokument angibt, welches partiell die Struktur des anzufragenden Dokumentes wiedergibt. Vereinfacht gesehen, versucht ein Anfrageprozessor alle möglichen (Teil-)Bäume der Datenquelle zu identifizieren, die diesem Pattern entsprechen. Der Benutzer muss also lediglich angeben, wie die Daten aussehen sollen, die er gerne befragen möchte. Dieser deklarative Ansatz, der auch in SQL und XML-QL (siehe [ADMFDF+98] und [SAPBDS00]) verfolgt wird, wird als Erleichterung für den Benutzer gesehen, da er lediglich angeben muss, "was" er will und nicht "wie" er die gesuchten Daten erhält.

Xcerpt ist eine regelbasierte Sprache: Ähnlich wie in Prolog werden Regeln verwendet, die nur dann ein Ergebnis konstruieren, wenn die Teile eines Rumpfes erfüllt werden können. Ansonsten scheitern sie.

Xcerpt verlangt (und ermöglicht) eine strikte Trennung zwischen Anfrage und Konstruktion: In Xcerpt wird konsequent die Idee verfolgt, dass Anfragen nur dazu verwendet werden, die Daten zu befragen und die möglichen Ergebnisse zu beschränken. In einem Konstruktionsteil werden die Ergebnisse aus der Anfrage wieder zusammengebaut oder neue Daten konstruiert. Es dürfen dort aber keine Einschränkungen auf die Anfrage vorgenommen werden.

Im folgenden werden die Sprachkonstrukte von Xcerpt anhand von einfachen Beispielen vorgestellt um einen tieferen Einblick und ein Gefühl für die Sprache zu vermitteln und die soeben vorgestellten Prinzipien näher zu erläutern.

Kapitel 2

Xcerpt: Eine Einführung in die Sprache

In diesem Kapitel wird ein Überblick über die Konstrukte von Xcerpt geliefert. Es wird allerdings keine formale Syntax eingeführt[1]. Ziel ist es, dem Leser ein Gefühl und das nötige Wissen für die Sprache zu geben, um die Anwendungsszenarien (Use Cases) im nächsten Kapitel nachvollziehen zu können. Hierzu werden einfache Ausschnitte und Beispiele aus den W3C Use Cases gegeben. Die kompletten Xcerpt-Programme sind allesamt im Anhang zu finden.

Die Beispieldaten:

Zur Erklärung wird das "XMP" Szenario aus den XQuery Use Cases genommen. In diesem Anwendungsszenario werden Anfragen auf zwei XML Dokumente mit unterschiedlicher Struktur gestellt, welche beide exemplarisch die Daten eines Online Buchladens beinhalten.

Die Datei "bib.xml"

```
<bib>
  <book year="1994">
    <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">
    <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>
```

[1] Die Einführung einer formale Syntax findet im Rahmen der Dissertation von Sebastian Schaffert statt. Die in dieser Arbeit vorgestellte Syntax wurde in internen Gesprächen mit Prof. Dr. François Bry und Sebastian Schaffert entwickelt

```
  <book year="2000">
    <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">
    <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>
```

Die Datei “reviews.xml”

```
<reviews>
  <entry>
    <title>Data on the Web</title>
    <price>34.95</price>
    <review>A very good discussion of semi-structured database
    systems and XML.</review>
  </entry>
  <entry>
    <title>Advanced Programming in the Unix environment</title>
    <price>65.95</price>
    <review>A clear and detailed discussion of UNIX
    programming.</review>
  </entry>
  <entry>
    <title>TCP/IP Illustrated</title>
    <price>65.95</price>
    <review>One of the best books on TCP/IP.</review>
  </entry>
</reviews>
```

2.1 Terme

Das elementarste Konstrukt der Sprache Xcerpt ist ein Term. Xcerpt selbst ist aus Termen aufgebaut und fragt wiederum Terme an. So werden z.B. semistrukturierte Datenbanken als Xcerpt-Datenterme repräsentiert. Xcerpt kennt drei Arten von Termen:

- Datenterme,
- Anfrageterme,
- und Konstruktionsterme.

Diese drei Arten ähneln sich in weiten Teilen. Ein Term besteht aus einem Bezeichner, welcher in XML einem Elementtag entspricht, gefolgt von einer Termspezifikation, die angibt, wie Subterme (in XML Unterelemente) behandelt werden sollen.

2.1.1 Geordnete/ungeordnete und partielle/totale Termspezifikation

Anfragen auf semistrukturierte Daten sind nicht unbedingt mit Anfragen auf voll strukturierte Daten vergleichbar. Voll strukturierte Dokumente entsprechen einem festgelegtem Schema. Für alle Elemente des Schemas muss es Einträge im Dokument geben. Elemente, die leer sind, beinhalten zumindest einen "Null"-Wert. Die Elementtags sind aber vorhanden. Optionalität und Kardinalität über das Auftreten von Elementen ist nicht möglich. Man spricht von homogenen Daten. Bei semistrukturierte Daten hingegen benötigt man nicht zwangsläufig ein Schema, auf das sich die Dokumentinstanzen beziehen. Angaben zur Optionalität und Kardinalität sind möglich, mit dem Effekt, dass leere Elemente im Dokument erst gar nicht auftreten müssen. Hier spricht man von heterogenen Daten.

Aus diesem Grund benötigt man Konstrukte in einer Anfragesprache, die vage und unvollständige Spezifikationen zulassen. Vagheit in der Breite kann durch unterschiedliche Klammerungsarten in den Termspezifikationen ausgedrückt werden. So kann man Terme als geordnet (z.B. `f[a,b,c]`) oder ungeordnet (z.B. `f{a,b,c}`) betrachten und wiederum die Eigenschaft der Ordnung zum Anfragen ausnutzen. Weiterhin ist es in Anfragen möglich Terme total zu spezifizieren (alle Subterme eines Datenterms müssen in der Anfrage angegeben werden) oder partiell (nur eine Teilmenge der Subterme wird in der Anfrage angegeben).

Eine Termspezifikation bezieht sich immer auf alle Subterme. So ist es nicht erlaubt, dass eine Teilmenge von Subtermen geordnet ist, während eine andere ungeordnet ist (z.B. `f{a,b,[c,d]}`). In diesem Beispiel müsste ein neuer Subterm eingeführt werden, der die Subterme `c` und `d` als Subterme eines geordneten Terms betrachtet (z.B. `f{a,b,z[c,d]}`). Terme ohne Subterme, die leeren XML-Elementen entsprechen, werden immer mit einfachen geschweiften Klammern spezifiziert (z.B. `f{}`).

2.1.2 Attribute

Attribute werden immer als ungeordnete Menge von Termen spezifiziert. Ansonsten wird aber dieselbe Term-Schreibweise wie bei "normalen" Termen verwendet. Als Schlüsselwort, welches der Xcerpt-Interpreter erkennt, benutzt man den Bezeichner `attributes`. Dieser Term muss immer das erste Kind des Vaterelements sein. Innerhalb dieses `attributes`-Terms, der geordnet spezifiziert sein muss, folgen die Terme, die die eigentlichen Attribute darstellen.

2.2 Datenterme (Data Terms

Xcerpt-Datenterme können, wie erwähnt, geordnet oder ungeordnet spezifiziert werden, ansonsten ähneln sie der Darstellungsart von semistrukturierten Daten. Aus logischer Sicht sind sie mit Atomen vergleichbar, wie sie beispielsweise in Prolog verwendet werden. Allerdings muss eine Xcerpt-Datenbank eine ausgezeichnete Wurzel besitzen, wie es auch bei XML Dokumenten verlangt wird. Es sei erwähnt, dass XML Dokumente in der Xcerpt-Darstellung immer geordnet sind.

Beispiel: Betrachte man ein Buch mit dem Titel "Data on the Web", welches von dem mehrere Autoren geschrieben wurde. Eine möglich XML Darstellung eines Datenterms ist:

```
<book>
  <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>
</title>
```

Die entsprechenden Xcerpt-Term-Syntax wäre:

```
book [
  title [ "Data on the Web" ],
  author [
    last [ "Abiteboul" ],
    first [ "Serge" ]
  ],
  author [
    last [ "Buneman" ],
    first [ "Peter" ]
  ],
  author [
    last [ "Suciu" ],
    first [ "Dan" ]
  ]
]
```

Schreibweise für Attribute: Attribute werden immer als ungeordnete Menge von Termen betrachtet. Im folgenden Beispiel wird das oben vorgestellte Buch um ein Attribut für ein Herausgabe-Datum und ein Attribut für die Auflage erweitert.

```
book [
  attributes {
    year [ "2000" ],
    editon [ "second" ]
  },
  title [ "Data on the Web" ],
  author [
```

```
    last [ "Abiteboul" ],
    first [ "Serge" ]
  ],
  author [
    last [ "Buneman" ],
    first [ "Peter" ]
  ],
  author [
    last [ "Suciu" ],
    first [ "Dan" ]
  ]
]
```

Referenzen: Um semistrukturierte Daten in einer Graphstruktur auszudrücken, kann man in XML den ID/IDREF Referenzmechanismus benutzen. Befragt man allerdings XML-Dokumente mit XSLT oder XQuery müssen die so definierten Eltern-Kind-Beziehungen explizit dereferenziert werden. In Xcerpt-Anfragen kann diese explizite Dereferenzierung über Vergleiche der gewünschten Attributswerte durchgeführt werden. Darüber hinaus gibt es in Xcerpt einen eigenen Referenzmechanismus über Xcerpt-Terme, der implizit durch die Sprache geleistet wird. Zur Beschreibung der Beziehungen gibt es zwei Konstrukte:

- `id @ t`: `id` steht für einen Namen des Terms `t`, mit dem der Term `t` identifiziert wird.
- `^id` steht für einen Verweis auf einen Term, der mit dem Identifikator `id` benannt ist. Bei einer Xcerpt-Anfrage ist diese Referenz schon implizit aufgelöst. Das Element, auf welches der Verweis sich bezieht, ist also an der Stelle des Verweises eingefügt.

Eine mögliche äquivalente Darstellung eine obigen Datenterme unter Anwendung des Referenzmechanismus wäre dann:

```
a1@ author { last ( "Abiteboul" }, first { "Serge" }},
a2@ author { last ( "Buneman" }, first { "Peter" }},
a3@ author { last ( "Suciu" }, first { "Dan" }},

book [
  attributes {
    year [ "2000" ],
    editon [ "second" ]
  },
  title [ "Data on the Web" ],
  ^a1,
  ^a2,
  ^a3
]
```

2.3 Anfrageterme (Query Terms)

Anfrageterme sind Pattern über Datenterme, vergleichbar mit Zielen aus der Logikprogrammierung. Man kann sie als unvollständige Muster der anzufragenden Daten sehen; sie beschreiben also eine ähnliche (aber nicht notwendigerweise gleiche) Struktur. Die Unvollständigkeit kann über zweierlei Arten ausgedrückt werden. In Breite durch partielle Termspezifikation (doppelt geschweifte bzw. eckige Klammern) und in der Tiefe durch das Xcerpt-Konstrukt `descendant`, welches im folgenden eingeführt wird.

Datenterme können nur über die Variablen in Anfragetermen gebunden werden. Das Ergebnis, das durch *Matchen* eines Anfrageterms auf einen Datenterm erzeugt wird, ist eine Menge von Substitutionen für die Variablen, die in einem Anfrageterm vorkommen. Jede Substitution stellt eine mögliche Bindung für die im Anfrageterm geschriebenen Variablen dar, abhängig von ihrem Auftreten in dem Anfragepattern.

Ein Anfrageterm besteht aus einem Bezeichner oder einer Variable, gefolgt von einer Termspezifikation, welche durch unterschiedliche Klammerungsarten ausgedrückt wird. Die Klammern beinhalten Subterme, die wiederum Anfrageterme sind.

Im folgenden werden anhand von Beispielen die unterschiedlichen Spezifikationsmöglichkeiten von Anfragetermen näher erläutert.

2.3.1 Ungeordnete und Partielle Termspezifikation

```
bib {{
  book {{}}
}}
```

Diese Anfrage matcht mit einem Datenterm, wenn es einen `bib`-Term gibt, der als Subterm ein Buch beinhaltet, welches beliebig viele (oder keine) Subterme besitzt (ausgedrückt durch die doppelt geschweifte Klammer).

2.3.2 Ungeordnete und totale Termspezifikation

```
bib {
  book {{}},
  article {}
}
```

In diesem Fall muss der Term `bib` genau einen `book`-Subterm beinhalten, welcher selbst wieder beliebige Subterme haben kann, und einen `article`-Subterm, der leer sein muss (ausgedrückt durch die einfach geschweiften Klammern ohne Inhalt). Die Reihenfolge von `book` und `article` ist nicht von Bedeutung.

2.3.3 Geordnete und partielle Termspezifikation

```
bib [[
  book {{}},
  article {}
]]
```

Diese Anfrage matcht mit einem Datenterm, wenn es einen `bib`-Term gibt, der als Subterm ein Buch beinhaltet, welches beliebig viele (oder keine) Subterme besitzt (ausgedrückt durch die doppelt geschweifte Klammer). Weiterhin muss es einen `article`-Term geben, der **nach** einem `book`-Term vorkommt (wegen der doppelten eckigen Klammern) und welcher keine Subterme beinhaltet (ausgedrückt durch die einfach geschweifte Klammer).

2.3.4 Geordnete und totale Termspezifikation

```
bib [
  book {{}},
  article {}
]
```

Der Datenterm mit dem Bezeichner `bib` muss genau einen leeren `article`-Subterm **nach** genau einem `book`-Subterm beinhalten, welcher selbst beliebig viele Subterme enthalten kann.

2.3.5 Literale

Bedingt durch die rekursive Definition von Termen ist ein Literal auch ein Anfrageterm. Literale sind die Blätter der Termstruktur.

2.3.6 Variablen in Anfragetermen

Anfrageterme können Variablen enthalten. Diese werden in Xcerpt über das Schlüsselwort `var`, gefolgt von einem Bezeichner deklariert.

Wenn ein Anfrageterm mit einen Datenterm matcht, werden alle in dem Anfrageterm deklarierten Variablen an alle passende Subterme gebunden. Sie werden also dazu benutzt, um Daten aus der Datenbank zu extrahieren und in einem Konstruktionsterm (siehe Abschnitt 2.3.9) verwendet, um eine neue Struktur wiederzugeben. Durch Verwendung unterschiedlicher Termspezifikationen können die möglichen Variablenbindungen eingesammelt werden. Verwendet man geordnete Anfrageterme, hängt die Bindung von der Position der Subterme ab, bei ungeordneter partieller Spezifikation werden alle Subterme gebunden.

Es gibt mehrere Arten von Xcerpt-Variablen in Anfragetermen:

- **Variablen ohne Einschränkungen** werden durch das Schlüsselwort `var` gefolgt von einem Variablennamen deklariert. Sie können mit jedem Subterm der Datenbank auf der angegebenen Ebene matchen.

  Beispiel:

  ```
  bib {{
    var Subterm
  }}
  ```

  Diese Anfrage über die oben vorgestellten Datenterme bindet alle Subterme von `bib{{}}`, also alle `book`-Terme an die Variable `Subterm`.

- **Variablen mit Einschränkungen** werden wie Variablen ohne Einschränkungen geschrieben, gefolgt von einem `as` (geschrieben "`->`") und einem Anfrageterm. Diese Variable kann dann nur mit Subtermen matchen, die dem Anfrageterm entsprechen.

  Beispiel:

  ```
  bib {{
    book {{
      var Subterm -> title{{}}
    }}
  }}
  ```

Hier werden alle `title`-Terme, die unter einem Buch auftreten an die Variable gebunden, aber keine anderen Terme wie z.B. `price`.

Eine Bedingung für Variablendeklaration ist allerdings die Wohlgeformtheit von Variablen. Dies bedeutet, dass keine zyklischen Terme über das "as" Konstrukt erlaubt sind, da eine Variable nicht an sich selbst gebunden werden kann.

- **Label Variablen:** Variablen sind nicht nur ein Mittel um Terme aufzusammeln, sondern ermöglichen auch Anfragen über unbekannte Strukturen. Wie gezeigt liefert eine Anfrage ohne Einschränkung alle Subterme, auch wenn man ihre Bezeichnungen nicht kennt.

  Darüber hinaus ist es auch möglich Strukturen nach ihrer Benennung zu befragen. Eine Variable wird hierbei an der Stelle eines Bezeichners deklariert. In den Kindtermen kann die Struktur weiter spezifiziert werden. Angenommen, man weiß einerseits, dass ein Dokument ein `bib`-Element als Wurzel besitzt, welches mehrere Bücher beinhaltet, und möchte alle Namen der direkten Kinder eines Buches herausfinden. Eine Anfrage über dem Dokument "bib.xml", die alle Kindelementnamen eines Buches selektiert lautet dann:

```
bib {{
  book {{
    var X {{ }}
}}
```

  Die Variable `X` wird an alle Bezeichner (aber nicht Subterme) gebunden, die der angegebenen Schachtelung entsprechen. Nach der Auswertung sind die möglichen Bindungen für die Variable `X` die Strings "title", "author", "publisher" und "price". (vgl. XML Dokument 2). Dieses Beispiel beschränkt sich auf die Bezeichnersuche der direkten Nachfolger des Elements `book`. Will man aber alle Bezeichnungen von Elementen erhalten, die in beliebiger Tiefe auftreten, benötigt man ein Konstrukt das weniger spezifizierte Anfrageformen ermöglicht, welches im nächsten Abschnitt vorgestellt wird.

2.3.7 Anfragen auf Attribute

Attribute können auch über partielle oder totale Spezifikationen befragt werden, allerdings ohne die Ordnung zu berücksichtigen. Die Attribute, des oben eingeführten Datenterms mit den Attributen `year` und `edition` können beispielsweise folgendermaßen angefragt werden.

Beispiel 1: Alle Attribute werden hier an die Variable `Attribute` gebunden.

```
bib{{
  book {{
    attributes {{
      var Attribute
    }}
  }}
}}
```

Beispiel 2: Die Variable `Attribute` wird an den Attributsterm `edition` gebunden, wenn es noch genau ein Attribut gibt, das den Namen `year` hat. Die Reihenfolge spielt dabei keine Rolle.

```
bib{{
  book {{
    attributes {
      var Attribute,
      year {{}}
    }
  }}
}}
```

2.3.8 Das Konstrukt "descendant"

Ein mächtiges Konstrukt, welches Anfragen noch allgemeiner schreiben lässt ist das "descendant"-Konstrukt (`desc`). Es ermöglicht, Subterme in der Datenbank zu suchen, die in beliebiger Tiefe auftreten. Die Suche beginnt von der Ebene aus, in der das Schlüsselwort `desc` auftritt.

Beispiel: Eine Anfrage, die alle Bücher selektiert, die in beliebiger Tiefe ein oder mehrere Elemente enthalten, die als Literal den String "Abiteboul" enthalten, lauten.

```
bib {{
  var Book -> book {{
    desc var X {"Abiteboul"}
  }}
}}
```

Diese Anfrage würde alle `book`-Terme an die Variable `Book` binden, welcher in beliebiger Tiefe einen Subterm besitzt, dessen Inhalt das Literal "Abiteboul" ist: In diesem Beispiel an das Buch mit dem Titel "Data on the Web". Die Variable `X` würde wieder an den String "last" gebunden werden, soll aber lediglich als Platzhalter für einen vom Namen nicht bekannten Term stehen. Ob man mit den Bindungen der Variable `X` neue Information erzeugt, oder diese nur als einen Platzhalter verwendet spielt nicht in dem Anfrageteil eine Rolle, sondern entscheidet sich in der Konstruktion von neuen Daten.

Im Moment ist das "descendant"-Konstrukt in seiner Tiefe nicht beschränkt. Allerdings ist eine Möglichkeit zur Beschränkung geplant, z.B. durch reguläre Pfadausdrücke oder durch die Angabe einer maximalen Tiefe. Es liegt nahe, dass diese Option vor allem in der Suche im Web nötig ist, z.B. beim Verfolgen von zyklischen Links.

2.3.9 Konstruktionsterme (Construct Terms)

Konstruktionsterme sind Ergebnispattern mit welchen man neue Terme bildet. Dies geschieht durch die explizite Angabe einer neuen Termstruktur. Anhand der Variablenbindungen aus den Anfragetermen wird die neue Struktur mit den Ergebnissen der Anfrage angereichert.

Ein Konstruktionsterm ist im Groben genauso definiert wie ein Datenterm. Er besteht also aus einem Bezeichner gefolgt von einer Termspezifikation, die entweder leer ist, oder ein oder mehrere Konstruktionsterme enthält. Ein Literal ist ein atomarer Konstruktionsterm. In Konstruktionstermen ist nur eine totale Termspezifikation erlaubt da das Ergebnis eines Konstruktionsterms die Menge der Datenterme erweitert, die selbst nur total spezifiziert sein kann. Im Gegensatz zu Datentermen sind Konstruktionsterme mit Variablen und speziellen Konstrukte angereichert, die dazu dienen, die möglichen Variablenbindungen auf unterschiedliche Weise auszugeben.

Beispiel: Folgender Konstruktionsterm kreiert ein Buch mit einem Author mit Namen "Schmidt" hinzu und fügt dieses zu der Menge der Datenterme hinzu.

```
book {
  author {
    last {"Schmidt"}
  }
}
```

2.3.9.1 Variablen

Variablen dienen in Konstruktionstermen als Platzhalter für die möglichen Bindungen, die sich aus der Anfrage ergeben. Es wird die selbe Syntax wie in den Anfragetermen verwendet, mit der Ausnahme, dass Variablenbeschränkungen nicht erlaubt sind. Dies würde keinen Sinn machen, da in Konstruktionstermen nur die Gestalt des Ergebnisses angegeben wird. Wegen der strikten Trennung von Anfrage und Konstruktion darf im Konstruktionsteil kein Einfluss auf die Anfrage genommen werden. Weiterhin darf eine Variable in einem Konstruktionsterm nur dann vorkommen, wenn sie auch in einem Anfrageterm angegeben wurde.

Beispiel: Die Bindung der Variable `Book` des obigen Anfragebeispiels in Abschnitt 2.3.8 kann mit folgendem Konstruktionsterm ausgegeben werden.

```
result {
  var Book
}
```

Das Ergebnis in Form eines Xcerpt-Datenterms ist dann ein Buch, welches einen Autor mit dem Namen "Abiteboul" hat.

```
result {
  book [
    attributes  {
      year [ "2000" ]
    },
    title [ "Data on the Web" ],
    author [
      last [ "Abiteboul" ],
      first [ "Serge" ]
    ],
    author [
      last [ "Buneman" ],
      first [ "Peter" ]
    ],
    author [
      last [ "Suciu" ],
      first [ "Dan" ]
    ],
    publisher [ "Morgan Kaufmann Publishers" ],
    price [ "39.95" ]
  ]
}
```

2.3.9.2 Sammelkonstrukte

Sammelkonstrukte, auch Aggregationskonstrukte genannt, werden verwendet, um mehrere mögliche Variablenbindungen auszugeben. Sie erscheinen immer vor der betroffenen Variable

in einem Konstruktionsterm.

- **Das Xcerpt-all-Konstrukt** liefert alle möglichen Variablenbindungen. So erzeugt der Konstruktionsterm,

  ```
  result {
    all var Book
  }
  ```

  bedingt durch einen Anfrageterm

  ```
  desc Book -> book {{}}
  ```

  alle `book`-Elemente mit seinen Subtermen, die in dem Dokument "bib.xml" von Kapitel 2 vorkommen.

- **Xcerpt-some-Konstrukt**: `some` gemeinsam mit einer ganzen Zahl `N`, geschrieben vor einer Variablen, liefert `1` bis `N` beliebige Bindungen dieser Variablen. Sammelkonstrukte liefern immer mindestens einen Wert einer Variable, da eine Variable immer mindestens eine Bindung haben muss, damit die Anfrage nicht scheitert.

  ```
  result {
    some 2 var Book
  }
  ```

  Dieser Konstruktionsterm, der ebenfalls die Variablenbindungen der obigen Anfrage zur Verfügung hat, würde zwei beliebige `book`-Terme liefern.

Die Ausgabe der Reihenfolge von Ergebnissen ist nichtdeterministisch. So erhält man mit `some N N` beliebige Terme der Ergebnismenge. In der Praxis liefert der Xcerpt-Prototyp die Instanzen allerdings in der Reihenfolge des Vorkommens.

2.3.9.3 Geschachtelte Sammelkonstrukte

Sammelkonstrukte können auch geschachtelt auftreten, um komplexere Ergebnisse auszugeben.

Beispiel: Betrachte man folgende Anfrage: *"Liste den Titel und alle Autoren für jedes Buch in der Bibliographie auf. Titel und Autoren sollen im Ergebnis innerhalb eines* `result`*-Elements gruppiert werden."*

Zunächst werden `title` und `author` in einem Anfrageterm an entsprechende Variablen gebunden.

Anfrageterm:

```
bib [[
  book[[
    var Title -> title {{}},
    var Author -> author {{}}
  ]]
]]
```

Der Konstruktionsterm, der alle möglichen `titel-author` Kombinationen liefert, wobei alle Autoren eines Buches ausgegeben werden müssen, lautet:

Konstruktionsterm I:

```
all result [
  var Title,
  all var Author
]
```

Dieser Konstruktionsterm liefert als Ergebnis alle Bücher mit allen Autoren eines Buches. Zu beachten ist, dass durch eine kleine Veränderung des Konstruktionsterms, aber ohne Veränderung des Anfrageterms eine ganz andere Anfrage formuliert wird. Betrachte man den Konstruktionsterm:

Konstruktionsterm II:

```
all result [
  all var Title,
  var Author
]
```

Man erhält nun alle Buchtitel, die ein Autor verfasst hat.

2.3.9.4 Sortierung

Die Konstrukte `all` und `some` geben auf Grund ihrer Definition die Instanzen in einer beliebigen Reihenfolge aus. In der Praxis liefert der Xcerpt-Prototyp die Instanzen allerdings in der Reihenfolge des Vorkommens. Häufig will man aber Daten nicht in der Reihenfolge ihres Auftretens ausgeben, sondern Ergebnisse nach unterschiedlichen Kriterien sortieren. Durch die Angabe des Konstrukts `order by` gefolgt von einer Liste von Variablen am Ende eines Konstruktionsterms kann eine Sortierung des Ergebnisses erreicht werden. Man kann also nur über Variablen sortieren, nicht über die Bezeichner eines Konstrukionsterms. Hierbei wird zuerst nach der ersten Variablen der Liste, dann nach der zweiten bis zur letzten sortiert. Weiterhin ist es möglich die Art der Sortierung zu bestimmen, was über die Schlüsselwörter `numeric` (z.B. “2” ist kleiner als “11”) und `literal` (z.B. “2” ist größer als “11”) geschieht. Aus der Annahme heraus, dass ein einzelner Wert nicht sortiert werden muss, kann allerdings nur sortiert werden, wenn ein Sammelkonstrukt vor der zur sortierenden Variable steht.

Beispiel: Der obige Anfrageterm in Abschnitt 2.3.9.3 wird etwas verändert, um die Ausgabe nach sinnvollen Kriterien zu sortieren.

```
bib [[
  book[[
    var Title -> title {{}},
    author {{
      last { var Last },
      first { var First }
    }}
  ]]
]]
```

Folgender Konstruktionsterm liefert alle Buchtitel mit all seinen Autoren, sortiert nach Nachname und dann nach Vorname.

```
results [
  all result [
    var Title,
    all  Author {
      last { var Last },
      first { var First }
    } order by [ Last, First ] literal
  ]
]
```

2.4 Programme

Xcerpt-Programme setzen sich aus Regeln ("rules") und Zielen ("goals") zusammen. Ein Programm muss wenigstens aus einer Regel oder einem Ziel bestehen. Ein Programm ist nicht explizit ausgezeichnet. Es gibt also keine syntaktischen Schlüsselwörter die den Anfang und das Ende eines Programms kennzeichnen. Alleine die Existenz von Regeln und Zielen ergeben ein Programm.

2.5 Regeln

Eine Regel ("rule") besteht aus einem Anfrageteil ("body" oder Rumpf) und einem Konstruktionsteil ("head" oder Kopf). Der Rumpf muss mindestens einen Anfrageterm beinhalten. Gibt es mehrere Anfrageterme in einem Rumpf, dann sind sie entweder durch Konjunktion (in Xcerpt mit `and{}`) oder Disjunktion (in Xcerpt mit `or{}`) verknüpft. Im Kopf kann nur genau ein Konstruktionsterm vorkommen. Eine Regel kann man als einen bedingten Ausdruck sehen. Sie ist erfolgreich, wenn es einen Datenterm gibt, der der Anfrage, die im Rumpf spezifiziert wird, entspricht und liefert dann auch ein Ergebnis in Form der Struktur, wie sie im Kopf angegebenen ist. Gibt es keinen solchen Datenterm, so scheitert sie.

Mit den Anfrageteilen können Webressourcen oder selbst wieder die Ergebnisse anderer Regeln befragt werden. Die im Anfrageteil gebundenen Variablen werden im Konstruktionsteil aufgelöst und dazu verwendet, neue Datenterme zu erzeugen. Eine einleuchtende Bedingung für die Verwendung von Variablen in Konstruktionstermen ist, dass diese auch in mindestens einem der Anfrageterme vorkommen muss, was im folgenden als Bereichsbeschränkung von Variablen bezeichnet wird.

Regeln werden durch das Schlüsselwort `CONSTRUCT` eingeleitet, welches den Beginn des Konstruktionsteils kennzeichnet, gefolgt von einem Konstruktionsterm. Das Schlüsselwort `FROM` leitet den Anfrageteil, gefolgt von einer Konjunktion oder Disjunktion von Anfragetermen. Das Ende einer Regel wird mit `END` gekennzeichnet.

2.5.1 Gerüst einer Regel:

```
CONSTRUCT
  <Konstrukt-Teil>
FROM
  <Query-Teil>
END
```

Beispiel: Betrachte man eine Anfrage, die für alle Bücher in "bib.xml" die Autoren oder Editoren gemeinsam mit dem Titel des Buches in einem `result` Element ausgeben soll.

```
CONSTRUCT
results {
  all result {
    var Title,
    all var Person
  }
}
FROM
or {
  bib [[
    book{{
      var Title -> title {{}},
      var Person -> author {{}}
    }}
  ]],
  bib [[
    book{{
      var Title -> title {{}},
      var Person -> editor {{}}
    }}
  ]]
}
END
```

Der Anfrageteil besteht aus einer Disjunktion von zwei Query Termen. Damit die Regel nicht scheitert, muss wenigstens eine der Anfragen erfüllt sein. Betrachte man dazu das XML Dokument "bib.xml" von Kapitel 2. Man sieht, dass einer der Anfrageterme immer erfüllt ist. Es wird also jeder Buchtitel im Konstruktionsteil erzeugt. Als Ergebnis erhält man jeden Buchtitel mit all seinen Autoren, und/oder seinen Editoren.

2.5.2 Rule Chaining

Wie oben erwähnt, können Regeln die Ergebnisse anderer Regeln befragen. Dies nennt man "Rule Chaining". Der Rumpf einer Regel wird also auf den Kopf einer anderen angewandt. Eine komplexe Regel kann somit durch mehrere einfachen Regeln ausgedrückt werden und somit die Überschaubarkeit von Programmen erhöhen. Außerdem dient Rule Chaining auch zum Zusammenfassen von Ergebnissen mehrerer Regeln.

Beispiel: Als Beispiel dient folgende Anfrage. *"Für jedes Buch mit Autoren, gib das Buch mit seinem Titel und seinen Autoren zurück. Für jedes Buch mit einem Editor erzeuge eine Referenz mit Buchtitel und der Organisationszugehörigkeit des Editors."*

Eine Lösungsmöglichkeit besteht darin, zwei Regeln zu verwenden um die unterschiedlichen Konstruktionsbedingungen zu erfüllen.

Die erste Regel liefert alle Bücher mit Titel und Autoren.

```
CONSTRUCT
  book [
    var Title,
    all var Author
  ]
FROM
  bib [[
    book [[
      var Title -> title {{}},
      var Author -> author {{}}
    ]]
  ]]
END
```

Die zweite Regel liefert alle Referenzen mit Titel und Organisationszugehörigkeit.

```
CONSTRUCT
  reference [
    var Title,
    affiliation [
      var Affiliation
    ]
  ]
FROM
  bib [[
    book [[
      var Title -> title {{}},
      editor [[
        affiliation [[
          var Affiliation
        ]]
      ]]
    ]]
  ]]
END
```

Die dritte Regel befragt die Ergebnisse der beiden oberen Regeln und gibt diese innerhalb eines `results` Elementes aus, indem man durch Disjunktion die gerade neu erzeugten Terme `book` und `reference` an die Variable `Book` bindet.

```
CONSTRUCT
  results [
    all var Book
  ]
FROM
  or {
    var Book -> book {{}},
    var Book -> reference {{}}
  }
END
```

2.6 Ziele

Ziele ("Goals") sind mit wenigen Ausnahmen den Regeln sehr ähnlich . Zum einem werden Ziele verwendet, um die Ergebnisse in einem bestimmten Format auszugeben und an unterschiedlichen Orten abzulegen, wie zum Beispiel in HTML-Format auf einer Web-Seite, lokalen

Dateien oder einfach in dem verwendeten Terminal. Dies wird durch die Spezifikation einer Ausgaberessource im Kopf eines Zieles (siehe Abschnitt 2.7) erreicht , was in einer Regel allerdings nicht möglich ist. Weiterhin können die Ergebnisse von Zielen nicht durch weitere Regeln oder Ziele befragt werden (deswegen der Name Ziel). Regeln müssen nicht notwendigerweise ausgewertet werden, sie scheitern einfach oder werden erst gar nicht ausgewählt. Ziele hingegen werden immer ausgewählt. Scheitert die Anwendung eines Zieles, dann ist das Ergebnis, ähnlich wie in Prolog, "no" oder "no results".

Wie erwähnt kann ein Programm mehrere Ziele beinhalten, die die Ergebnisse in unterschiedlichen Formaten ausgeben. Bedingt durch die Tatsache, dass die Ziele möglicherweise gemeinsame Zwischenergebnisse von Regeln befragen können, verspricht man sich hier einen Ansatzpunkt für effiziente Berechnung.

Die Ausgabe eines Ziels muss wiederum semistrukturiert sein und es muss also ein Baum mit einem ausgezeichneten Wurzelterm geliefert werden. Würde ein Konstruktionsteil eines Zieles mehrere Ergebnisterme auf der Wurzelebene liefern, würde nur der erste ausgegeben werden. Dieses Problem kann allerdings leicht durch das Angeben eines übergeordneten Wurzelelements vermieden werden. Enthält ein Programm aber zwei oder mehr Ziele, muss darauf geachtet werden, dass diese nicht die selben Ausgabespezifikationen enthalten dürfen.

Die dritte Regel des Programms von Abschnitt 2.5.2 kann man auch als Ziel schreiben, indem man das Schlüsselwort `CONSTRUCT` mit `GOAL` ersetzt. Das Ergebnis kann dann nicht weiter angefragt werden.

2.7 Spezifikation von anzufragenden und auszugebenden Ressourcen

Ressourcen Angaben werden benötigt um auf externe Datenquellen zuzugreifen und neue zu erstellen. Ressourcenangaben beziehen sich auf semistrukturierte Daten, wie sie in Xcerpt, XML [xml00], HTML[DRALHIJ99], LISP u.v.m. vorkommen. Eine Ressourcenspezifikation besteht aus der vom W3C vorgeschlagenen IRI-Spezifikation[2] und eventuell einer Angabe des Typs der Datenquelle. Man unterscheidet zwischen Eingabe- und Ausgaberessourcen.

2.7.1 Eingaberessourcen

Eingaberessourcen dienen zum Befragen von externen Datenquellen und werden durch den Xcerpt-Term `in{}` eingeleitet, gefolgt von einem `resource`-Term und anschließend von Regeln oder Zielen, oder Anfrageteilen. Eine Eingaberessource beschreibt einen Verweis auf eine Datei einer Web-Seite oder auf eine lokale Datei, gepaart mit einer Typbeschreibung des Datenformats . Die nötigen Angaben werden in einem `resource`-Term spezifiziert, der eine Pfad auf die Datenquelle und optional eine Angabe des Typs der Datenquelle (z.B XML, HTML, Xcerpt, etc.) beinhaltet. Eingabespezifikationen können an mehreren Stellen vorkommen:

- Sie können ein Programm,
- mehrere Regeln oder Ziele umschließen,

[2] http://www.w3.org/International/O-URL-and-ident.html

- oder nur Anfrageteile beinhalten.

In Xcerpt-Programmen sind mehrere Ressourcenangaben möglich. Zum einen sind Konjunktionen und Disjunktionen von Ressourcen möglich (ausgedrückt durch die Xcerpt-Terme `and{}` und `or{}`). In diesem Fall werden die Anfragen nacheinander auf die Ressourcen angewandt und die Ergebnisse werden als Konjunktion oder Disjunktion ausgegeben. Dies ermöglicht, dass man nur eine Anfrage benötigt, um auf unterschiedlichen Datenquellen Anfragen zu stellen.

Beispiel: Eingabespezifikation mit Konjunktion über zwei Quelldokumente

```
in {
  and {
    resource{"file:dok1.xml","xml"},
    resource{"file:dok2.xml","xml"}
  },
  <ANFRAGETEIL>
}
```

Zum Anderen sind geschachtelte Ressourcenangaben möglich. Hierbei gilt das Prinzip der Überschattung. Die innerste Spezifikation gilt.

2.7.2 Ausgaberessourcen

Ausgaberessourcen dienen zur Ablage der Anfrageergebnisse in bestimmten Dateien und an bestimmten "Orten" und werden über den Term `out{}` spezifiziert, gefolgt von einem `resource`-Term und einem Konstruktionsteil eines Zieles oder mehreren Zielen. Über eine Ausgaberessource wird der Pfad angegeben, wo das Ergebnis gespeichert werden soll (z.B. auf einer Web-Seite, auf der lokalen Festplatte oder im Standartoutput des verwendeten Terminals) und in welchem Format (z.B. Xcerpt, HTML, XML etc.) das Ergebnis abgespeichert werden soll. Ausgaberessourcen dürfen nur in Verbindung mit Zielen verwendet werden. Hierbei umfassen sie entweder mehrere Ziele, oder sie treten vor dem Konstruktionsteil eines Zieles auf.

Beispiel: Dieses Beispiel ist eine Erweiterung des Programms von Abschnitt 2.5.2, welches um Spezifikationen von Eingabe- und Ausgaberessourcen bereichert ist.

```
GOAL
  out {
    resource { "stdout:", "html" },
    results [ all var Book ]
  }
FROM
  or {
    var Book -> book {{}},
    var Book -> reference {{}}
  }
END

CONSTRUCT
  book [
    var Title,
    all var Author
  ]
```

```
FROM
  in {
    resource { "file:xmp-bib.xml", "xml" },
    bib [[
      book [[
        var Title -> title {{}},
        var Author -> author {{}}
      ]]
    ]]
  }
END

CONSTRUCT
  reference [
    var Title,
    affiliation [ var Affiliation ]
  ]
FROM
  in {
    resource { "file:xmp-bib.xml", "xml" },
    bib [[
      book [[
        var Title -> title {{}},
        editor [[
          affiliation [[ var Affiliation ]]
        ]]
      ]]
    ]]
  }
END
```

2.8 Weitere Spezialkonstrukte

Die bisher vorgestellte Xcerpt-Basissprache ist nicht mächtig genug um auch komplexe Anfragen zu stellen.

Man ist zwar in der Lage über geordneten Termspezifikation positionelle Anfragen zu stellen wie z.B. in einem medizinischem Bericht, der in zeitlicher Reihenfolge "Einschnitte" und "Aktionen" beinhaltet. Eine mögliche Anfrage ist: *"Suche alle Aktionen, die nach mindestens zwei Einschnitten ausgeführt wurden"*.

```
desc var X [[
    incision {{}},
    incision {{}},
    desc var Action -> action {{}}
  ]]
```

Allerdings ist dies relativ aufwändig, da man das Pattern recht ausführlich spezifizieren muss, ohne dass die zusätzliche Information "interessant" wäre.

Würde man eine "Aktion" suchen, die nach mindestens zwölf "Einschnitten" ausgeführt wurde, würde der Anfrageterm ausarten. Daher wird in diesem Fall eine Funktion benötigt, mit der man einfach auf Positionen zugreifen kann.

Eines der gesteckten Ziele dieser Diplomarbeit ist auch, solche für komplexe Anfragen nötigen Konstrukte anhand der W3C Use Cases zu identifizieren und in eine sinnvollen Schreibweise

zu definieren.

2.8.1 Funktionen, Operationen und Constraints

Es lassen sich eine Vielzahl von wünschenswerten Funktionen finden, die eine Anfragesprache beinhalten soll. So benötigt man arithmetische Funktionen auf unterschiedlichen Typen wie die Addition, Konkatenation von Strings und Aggregationen wie den Durchschnittswert einer Menge von Ergebniswerten. Weiterhin sollte es auch möglich sein, die Anfrageterme um Bedingungen zu erweitern um die Patterns auf einfache Weise "enger" zu spezifizieren.

Zu beachten ist aber die Einschränkung, dass neue Daten nur in Konstruktionsteilen erzeugt werden dürfen, während der Anfrageteil nur als eine Schablone gesehen wird, die man auf die Daten "legt". Die Verwendung von Funktionen und Aggregationen ist daher auf die Köpfe von Regeln und Zielen beschränkt, Bedingungen auf die Rümpfe. Dies lässt, wie im folgenden gezeigt wird, manche Xcerpt-Programme umständlich erscheinen, ist aber nötig um die strikte Trennung zwischen Anfrage und Konstruktion zu gewährleisten.

In dieser Arbeit werden diejenigen Funktionen und Aggregationen vorgestellt, die für die Verwirklichung der Anwendungsszenarien mit Xcerpt essentiell sind. Weitere Konstrukte werden während der Entwicklungsphase von Xcerpt noch hinzukommen.

2.8.1.1 Basistypen

Um Berechnungen über Subterme und Variablen durchführen zu können, müssen diese mit einem Typ versehen sein. So gibt es Operatoren, wie z.B. "+", die auf Integer, Float aber auch Strings angewandt werden können und jeweils unterschiedlich ausgewertet werden müssen. Der Interpreter muss frühzeitig erkennen, ob die Operationen auf den gegebenen Werten auch typkonform sind.

Im Moment sind in Xcerpt nur die wichtigsten Basistypen definiert. Allerdings ist ein komplexeres Typsystem in Arbeit, welches auch mit Schema-Informationen für Terme, mit Typinferenz und mit Typüberprüfung umgehen kann. Weitere Forschungsthemen für Xcerpt am Lehrstuhls für Programmier- und Modellierungssprachen der Ludwigs Maximilian Universität München[3] sind Typen über Ort und Zeit.

Als Basistypen stehen momentan in Xcerpt `integer`, `float`, `string` und `term` zur Verfügung.

Während die ersten drei Typen hinlänglich bekannt sind, beschreibt der Typ `term` die Menge aller Datenterme. Funktionen auf diesen Typ können Aussagen über Termeigenschaften wie Eltern-Kind Beziehungen liefern. Sie können aber nicht den Inhalt berücksichtigen. Subterme werden mit Typdeklarationen versehen, indem zwei Doppelpunkte (::) am Ende des Subterms geschrieben werden, gefolgt von einem Typbezeichner.

Beispiel für einen getypten Datenterm:

```
price{4.32::float}
```

Beispiel für eine getypte Variable in einem Anfrageterm:

```
price{var Price::float}
```

[3] http://www.pms.informatik.uni-muenchen.de

Fehlt eine Typdeklaration, so ist der Subterm automatisch vom Typ `term`.

2.8.1.2 Funktionen und Operationen

Funktionen und Operationen benötigen ein oder mehrere Argumente von vorgegebenen Typen als Eingabe und liefern ein Ergebnis mit vorgegebenen Typ. Folgende Operationen und Funktionen die in Xcerpt vorhanden sind werden für die Anwendungsszenarien benötigt

Funktionen zur Typumwandlung

- `toStr(a)`: Das Argument `a` beliebigen Typs wird zu einem String umgewandelt, wobei alle Sonderzeichen und Inhalte des Markups in die neue Zeichenfolge mit aufgenommen werden.
- `toText(a)`: Das Argument a beliebigen Typs wird zu einem String umgewandelt, wobei alle Sonderzeichen und Inhalte des Markups eliminiert werden.
- `toInt(a)`: Das Argument `a` beliebigen Typs wird zu einem Integer umgewandelt.
- `toFloat(a)`: Das Argument `a` beliebigen Typs wird zu einem Float umgewandelt.

Arithmetische Funktionen

Es gibt zwei verschiedene Schreibweisen bei der Benutzung dieser Funktionen. Einerseits eine generische Präfixschreibweise (z.B. `add(1, 2)`) und andererseits eine spezifische Infixschreibweise (z.B. (`1+2`). `t1` und `t2` stehen hierbei für Terme mit numerischem Typ.

- `add(t1,t2)` oder `t1+t2`
- `sub(t1,t2)` oder `t1-t2`
- `mult(t1,t2)` oder `t1*t2`
- `div(t1,t2)` oder `t1/t2`

Beispiel: Innerhalb eines Konstruktionsterms wird die die Differenz zweier Variablen, die an Preise gebunden sind, berechnet.

```
price_dif {
  var PriceA::float - var PriceB::float
}
```

String-Funktionen

- `concat(s1, s2)`: Fügt die beiden Strings `s1` und `s2` zusammen.

All diese Funktionen besitzen vorerst nur Konstruktionsterme als Argumente, da sie neue Ergebnisse kreieren und somit klar unter die Kategorie “Konstruktion” fallen. Da aber manche Anfragen Einschränkungen fordern, die nur arithmetisch zu lösen sind, wird im Abschnitt 2.8.1.4 eine Aufweichung der strikten Trennung vorgestellt.

2.8.1.3 Aggregationsfunktionen

Während Funktionen eine feste Stelligkeit von Argumenten verlangen, können Aggregationen eine nicht vorbestimmte Anzahl von Argumenten verarbeiten. Sie werden zumeist im Zusammenhang mit den Konstrukten `all` und `some` verwendet, die alle möglichen Variablenbindungen einer Anfrage aufsammeln. Folgende Aggregationen stehen zur Verfügung:

- `count`: Berechnet die Anzahl der Argumente. Die Argumente sind vom Typ `term`.
- `sum`: Summiert alle Argumente, die von Typ `integer` oder `float` sein müssen.
- `avg`: Berechnet den Durchschnitt aller Argumente, die von Typ `integer` oder `float` sein müssen.
- `min`: Liefert das Minimum aller Argumente, die von Typ `integer` oder `float` sein müssen.
- `max`: Liefert das Maximum aller Argumente, die von Typ `integer` oder `float` sein müssen.
- `join`: Vereinigt alle Argumente, die vom Typ `String` sein müssen, zu einem String.
- `first N`: Liefert die ersten `N` Argument einer Liste von Termen (Typ `term`).
- `drop N`: Schneidet die ersten `N` Argumente eine Liste von Termen ab.
- `reverse`: Liefert die Liste von Termen in umgekehrter Reihenfolge.

Beispiel für eine Aggregation mit count:

Dieses Beispiel berechnet die Anzahl aller Bücher, die in dem Dokument "bib.xml" enthalten sind, bis auf eine Ausnahme: Gibt es kein Buch in dem Quelldokument, scheitert die Anfrage. Das gewünschte Ergebnis wäre aber die Zahl "0". In Abschnitt 2.8.4 wird erklärt, wie solche Grenzfälle in Verbindung mit Aggregationen umgangen werden können.

```
CONSTRUCT
book_count {
  count (all var Book)
}
FROM
  desc var Book -> book {{}}
END
```

Beispiel mit drop und first:

Folgende Regel liefert den dritten Autoren des Buchs "Data on the Web", also den Autor "Dan Suciu".

```
CONSTRUCT
  third_author {
    first (drop 2 (all var Author))
}
FROM
  bib [[
    book[[
```

```
      title {"Data on the Web"},
      var Author -> author{{}}
    ]]
  ]]
END
```

Wie auch die Funktionen im obigen Kapitel sind Aggregationen nur über Konstruktionsterme erlaubt.

2.8.1.4 Constraints

Xcerpt verfolgt den Ansatz der patternbasierten Anfrage. Viele Bedingungen können über "eingeschränkte Variablen" und partielle Pattern ausgedrückt werden. Hierbei werden aber nur Besonderheiten der Struktur wie Eltern-Kind- und Geschwister-Beziehungen berücksichtigt. Einschränkungen, wie sie in Anfragen der Gestalt *"liefere alle Bücher, die nach 1993 erschienen sind"* nötig sind, können nicht ausgedrückt werden. Eine so genannte "Condition Box" wird daher verwendet. Diese kann an alle Anfrageterme und Subterme hinzugefügt werden. Bedingungen können nur über Variablen gefasst werden, die sich in dem Anfrageterm befinden, an welchem die "Condition Box" angefügt ist. Hierbei wird jede Bindung der Variable einzeln betrachtet. Sammelkonstrukte wie `all` und `some` sind aus diesem Grund nicht erlaubt. Aggregationsfunktionen können daher auch in der "Condition Box" nicht verwendet werden. Aus diesem Grund müssen in Programmen mit Aggregation häufig zusätzliche Regeln geschrieben werden um die Datenterme für weitere Regeln vorzubereiten.

Bedingungen werden durch das Schlüsselwort `where` eingeleitet, gefolgt von einem Condition-Ausdruck, der nur Wahrheitswerte liefert. Ein Condition-Ausdruck kann wiederum aus der Konjunktion oder Disjunktion von Condition-Ausdrücken bestehen. Die Wahrheitswerte erlangt man über Boolesche Funktionen, die im folgenden sind:

Boolesche Funktionen:

- `t1 eq t1`: Das linke Argument ist identisch mit dem rechten.
- `t1 ne t2`: Das linke Argument ist ungleich dem rechten Argument.
- `t1 lt t2`: Das linke Argument ist kleiner als das rechte.
- `t1 gt t2`: Das linke Argument ist größer als das rechte.
- `t1 leq t2`: Das linke Argument ist kleiner oder gleich als das rechte.
- `t1 geq t2`: Das linke Argument ist größer oder gleich als das rechte.
- `t1 substring t2`: Das linke Argument ist ein Substring des rechten Arguments.

Beispiel: In dem folgenden Anfrageterm werden alle Preise an die Variable `price` gebunden, wenn der Wert inklusive Mehrwertsteuer von 16% nicht über "40" liegen. Jedes vorkommen der Variable `price` muss mit einer Typangabe versehen werden.

```
bib {{
  desc price {var Price::float}
}}
where { add (var Price::float, mult( 0.16, var Price::float))  geq 40 }
```

2.8.2 Positionsangaben in Anfragen

Viele Anwendungsfälle beinhalten Anfragen, die Selektionen von Subtermen an bestimmten Positionen erfordern (vgl. hierzu die Anfrage am Anfang von Abschnitt 2.8).

Dieser Art der Patternanfrage würde aber ausarten, wenn es sehr viel Subterme geben würde und man in dieser geordneten Sequenz Terme selektieren möchte, die zum Ende dieser Sequenz hin auftreten.

Xcerpt bietet ein Konstrukt an, was die Selektion über Positionen ermöglicht. Hierbei wird vor einem Anfrageterm das Schlüsselwort `position` geschrieben, gefolgt von einer Positionsspezifikation.

`position <pos-spec>`

Eine Positionsspezifikation kann sein:

- Eine positive Zahl: Spezifiziert die Position des zu matchenden Subterms , wobei 1 die Position des ersten Subterm darstellt.
- Eine negative Zahl: Spezifiziert die Position des zu matchenden Subterms, wobei -1 die Position des letzten Subterm darstellt.
- Eine Variable: Wird eine Variable angegeben, matcht die Anfrage mit alle Subtermen und bindet die Variable an die Position des Subterms in Form einer positiven Zahl.

Eine Voraussetzung für die Verwendung von `position` ist, dass die angefragten Terme in einer Datenbank über Ordnung spezifiziert sein müssen. Die mit einem `position` versehenen Anfrageterme können innerhalb einer partiellen ungeordneten oder einer partiellen geordneter Termspezifikation stehen, aber logischerweise nicht innerhalb einer totalen Termspezifikation; in totalen Termspezifikationen müssen alle Subterme explizit angegeben werden.

Häufig bieten sich aber zur positionelle Anfrage auch die Aggregationen `first N`, `drop N` und `reverse` an (siehe Anfrage in Abschnitt 2.8.1.3). Das Xcerpt-Konstrukt `position` bezieht sich nämlich nur auf die Position von Subtermen im Allgemeinen, kann aber nicht in Verbindung mit der Benennung der Subterme angewandt werden. So liefert der Anfrageterm

```
desc book [[
  position 3 var X -> author{{}}
]]
```

nicht den dritten `author`-Term, sondern bindet den Term an die Variable `X` genau dann, wenn ein `author`-Term der dritte Subterm von `book` ist. Andernfalls scheitert die Anfrage. In diesem Beispiel würde der Anfrageterm den `author`-Term mit “Serge Abiteboul” an die Variable binden, da dieser Term als einziger dem Pattern, angewandt auf das Beispieldokument, entspricht. Will man hingegen den dritten `author`-Subterm, so müssen mit einer Query zuerst alle `author`-Subterme unter Berücksichtigung der Ordnung aufgesammelt und nacheinander in einem Konstruktionsterm wieder ausgegeben werden. Zusätzlich muss darauf geachtet werden, dass die Beziehung zwischen Buch und dessen Autoren erhalten bleibt. Deswegen wird der Buchbezeichner mit selektiert und zur Gruppierung im Konstruktionsteil verwendet.

```
CONSTRUCT
books {
  all var Label [
    all var Author
  ]
}
FROM
var Label [
  var Author -> author{{}}
]
END
```

Mit einer neuen Regel wird dann das dritte `author`-Element selektiert und ausgegeben.

```
CONSTRUCT
third_author [
  var X
]
FROM
books {{
  book [[
    position 3 var X
  ]]
}}
END
```

2.8.3 Negation

2.8.3.1 Subterm Negation

Eine wichtige Erweiterung für Xcerpt ist Negation. Sehr häufig will man Terme selektieren, die bestimmte Subterme nicht beinhalten, was hier im folgenden als "Subterm Negation" bezeichnet wird. In Xcerpt wird diese Verneinung mit dem Schlüsselwort `without` vor einem Anfrageterm ausgedrückt.

So liefert die folgende Regel alle Bücher, die nicht von "Addison-Wesley" verlegt wurden.

```
CONSTRUCT
 result {
    all var Book
  }
FROM
  in {
    resource{"file:bib.xml","xml"},
    desc bar Book -> book {{
      without publisher { "Addison-Wesley"}
    }}
  }
END
```

Darüber hinaus ist `without` ein wichtiges Konstrukt, um in Anfragetermen Bedingungen auszudrücken. Somit kann man Regeln schreiben, die sich gegenseitig ausschließen. Betrachte man nochmals das Beispiel in Abschnitt 2.5.2, in welchem man für jedes `title`-Element alle Autoren und Editoren finden sollte. Wenn ein Buch einen Autor enthält, sollte ein anderer Ergebnisterm erzeugt werden, als wenn es einen Editor enthält. In dem vorgestelltem Datentermen kommen nur Bücher die entweder Autoren oder Editoren enthalten. Angenommen,

die Datenbank enthält aber noch zwei weitere Bücher, eines ohne Autoren und Editoren, eines mit Autoren und Editoren:

Erweiterung des Dokuments um Datenterme:

```
book [
  title {"I wrote myself"}
],
book [
  title {"Impossible Objects as Nonsense Sentences"},
  author [
    first {"D.A."},
    last {"Huffman"}
  ],
  editor [
    first {"Bernard"},
    last {"Meltzer"}
  ],
  editor [
    first {"Donald"},
    last {"Michie"}
  ],
  publisher {"American Elsevier"}
]
```

Bei der Verwendung der Regel des Beispiels in Abschnitt 2.5.2 würden das Buch mit Autoren und Editoren doppelt ausgegeben werden, da jede Regel auf den neuen Datenterm angewandt werden kann. Das Buch "i wrote myself" würde gar nicht im Ergebnis enthalten sein, da keine Anfragepattern auf den Datenterm "passt".

Mit `without` kann man jedoch Regeln schreiben, die sich gegenseitig ausschließen, sodass immer nur genau ein Anfrageteil mit einem Datenterm matchen kann. Angenommen, man möchte für jedes Buch Titel, Autoren und Editoren zurückgeben und wenn es keinen Autor oder Editor gibt, den Term `no_authors{}` und `no_editors{}` hinzufügen. Um alle Kombination der (Nicht)/Existenz von Autoren und Editoren zu berücksichtigen, werden folgende vier Anfrageterme (für jede Regel einer) benötigt. Es werden nur die Anfrageterme aufgezeigt, die die nötigen Bedingungen ausdrücken. Eine Vereinfachung des Programms, welches mit nur einer Regel auskommt, wird in Abschnitt 2.8.4 vorgestellt.

Anfrageterm 1:

```
CONSTRUCT
results {
  all results {
    var Title,
    all var Author,
    no_editors{}
  }
}
FROM
desc -> book {{
  var Title -> title {{}},
  var Author -> author {{}},
  without editor {{}}
}}
END
```

Anfrageterm 2:

```
CONSTRUCT
results {
  all results {
    var Title,
    no_Authors{},
    all var Editor
  }
}
FROM
desc -> book {{
  var Title -> title {{}},
  without author {{}},
  var Editor -> editor {{}}
}}
```

Anfrageterm 3:

```
CONSTRUCT
results {
  all results {
    var Title,
    all var Author,
    all var Editor
  }
}
FROM
desc -> book {{
  var Title -> title {{}},
  var Author -> author {{}},
  var Editor -> editor {{}}
}}
```

Anfrageterm 4: Dieser Term wird nur benötigt, wenn alle Titel gefunden werden sollen, egal ob es einen Editor oder Autor gibt.

```
CONSTRUCT
results {
  all results {
    var Title,
    no_authors{},
    no_editors{}
  }
}
FROM
desc -> book {{
  var Title -> title {{}},
  without author {{}},
  without editor {{}}
}}
```

2.8.3.2 Query Negation

Manchmal kann es auch erwünscht sein eine kompletten Anfrageteil zu negieren, wie es auch in logischen Programmiersprachen wie Prolog möglich ist. In diesem Fall spricht man von "Query Negation". In Xcerpt wird dies durch ein `not` vor einem Anfrageteil ausgedrückt.

```
not <query-teil>
```

Der Unterschied zwischen den beiden Arten der Negation ist, dass eine Subterm Negation eine existenzielle Negation ist. Die Regel wird ausgeführt, wenn es wenigstens einen Datenterm gibt, der mit dem angegebenen Pattern matcht (also den negierten Subterm nicht enthält). Im Gegensatz dazu ist die Query Negation universell. Eine Regel ist nur dann erfolgreich, wenn es keinen Datenterm gibt, der der Anfrage entspricht.

Die "Query Negation" erscheint vor allem in Anfragen, die sich auf Algorithmen aus Logikprogrammierung notwendig zu sein. Häufig werden Regeln aus der Logik nur dann erfüllt, wenn ein Prädikat im Rumpf einer Regel nicht erfüllt wird (z.B. `p <- not(q),r`). Ein Beispiel für die "Query Negation" ist in Abschnitt 3.1.4.3 zu finden.

2.8.4 Optionalität von Subtermen

Nicht selten will man Terme selektieren, abhängig davon ob sie existieren oder nicht. Dies kann aber sehr aufwendig werden, wenn man alle Fälle von Existenz und Nichtvorhandensein von Subtermen beachten will. So müsste man alle Möglichkeiten für optional vorhandene Subterme in zusätzlichen Regeln ausdrücken, die sich wiederum durch Formulierung von Bedingungen gegenseitig ausschließen müssen.

Aus diesem Grund gibt es in Xcerpt ein Konstrukt, das es ermöglicht, einen Subterm als `optional` zu kennzeichnen. Ein Pattern matcht dann auch in dem Fall, in dem der betroffene Subterm nicht vorhanden ist.

`optional <query-term>`

Enthält ein mit `optional` versehener Subterm Variablen, so kann es vorkommen, dass eine Variable nicht gebunden werden kann. Aus diesem Grund müssen diese Variablen in Konstruktionstermen ebenfalls mit `optional` gekennzeichnet werden.

`optional var <identifier>`

Wenn es nun keine Variablenbindung gibt, wird auch kein Ergebniswert ausgegeben. Da dies ist aber nicht immer erwünscht ist gibt es die Möglichkeit, einen Standardwert anzugeben.

optional var <identifiere> with default <cons-term>

Beispiel: Als Beispiel wird das Szenario, welches in Abschnitt 2.5.2 eingeführt wurde, weiterentwickelt. In Abschnitt 2.8.3 wurde eine Möglichkeit aufgezeigt, durch sich gegenseitig ausschließende Regeln unterschiedliche Ergebnisse auszugeben, abhängig von der Gestalt der Datenterme. Allerdings wird das Programm durch die vielen Regeln unübersichtlich. Mit `optional` hat man die Möglichkeit mit einer einzigen Regel das selbe Ergebnis zu erzielen. Die folgende Regel liefert alle Titel mit Autoren und/oder Herausgebern. Wenn es keine Autoren oder Herausgeber gibt, soll ein leeres Element mit Label `no_authors` bzw. `no_editors` erzeugt werden.

```
CONSTRUCT
results {
  all results {
    var Title,
    all optional var Author with default no_authors{},
    all optional var Editor with default no_editors{}
  }
```

```
}
FROM
  bib {{
    book {{
      var Title -> title {{}},
      optional var Author -> author {{}},
      optional var Editor -> editor {{}},
    }}
  }}
END
```

2.8.5 Bedingte Anfragekontrolle

Beim Programmieren von Xcerpt-Anfragen verfolgt man oft folgenden Ansatz: "Wenn es einen Datenterm gibt, der einem Pattern entspricht, dann konstruiere einen neuen Datenterm. Wenn es aber einen Datenterm gibt, der einem **anderen** Pattern entspricht, dann konstruiere etwas anderes". Solche Art der Anfragen können nur über mehrere Regeln ausgedrückt werden, wobei beim Programmieren darauf geachtet werden muss, ob sich die Anfrageteile der unterschiedlichen Regeln gegenseitig ausschließen müssen oder überschneiden dürfen. Im ersten Fall kann nur eine Regel angewandt werden, im zweiten beide.

Hinter der bedingten Anfragekontrolle steht allerdings eine andere Idee. So erscheint es nützlich, in einem Anfrageteil zuerst eine "Stichprobe" auf Datenquellen auszuführen, bevor man weitere Datenquellen betrachtet. Zum Beispiel könnte man nach bestimmten Referenzen in einem Dokument suchen und, falls es welche gibt, in dieser Datenquelle nach gewünschten Subtermen suchen. Der Vorteil dabei ist, dass somit von vornherein manche Dokumente erst gar nicht geladen werden müssen. Folgende Einschränkung ist zu beachten: Da eine Negation von Variablenbindungen nicht definiert ist, darf im `ELSE`-Teil keine Variable auftreten, die im `IF`-Teil geschrieben wurde.

Beispiel: Das folgende Beispiel zeigt einen Anfrageteil mit "bedingter Anfrage" über die zwei Dokumente "bib.xml" und "reviews.xml" von Kapitel 2.

```
FROM
in {
  resource {"file:bib.xml","xml"},
  IF
    bib {{
      desc var Title -> title {"Data on the Web"}
    }}
  THEN
    in {
      resource {"file:reviews.xml","xml"},
      reviews {{
        entry {{
          var Title,
          var Review -> review{{}}
        }}
      }}
    }
  ELSE
    var Bib -> bib {{}}
}
```

In dem `IF`-Teil wird versucht den Titel "Data on the Web" im Dokument "bib.xml" zu finden. Ist die Anfrage erfolgreich, versucht man den selben Titel im Dokument "reviews.xml"

zu finden. Ist dieser vorhanden, soll der `review`-Term an eine Variable gebunden werden. Scheitert der Anfrageterm im `IF`-Teil, oder der Query-Term im `THEN`-Teil, wird das gesamte Inhalt von "bib.xml" an die Variable `Bib` gebunden.

2.8.6 Reguläre Ausdrücke

Mit den bisher vorgestellten Konstrukten kann man mit Xcerpt-Programmen Anfragen auf die Datenbank Struktur stellen und lediglich den kompletten Inhalt der Blätter eines Terms selektieren. Dies mag in datenzentrierten Dokumentstrukturen ausreichend sein, in textzentrierten sind solch beschränkten Anfragen aber zu schwach. Ein Konstrukt, welches sich zur Befragung von Text anbietet sind reguläre Ausdrücke, wie sie aus Programmiersprachen wie Perl[4], Python[5] und Java[6] bekannt sind. Diese bieten sich für Xcerpt besonders an, da sie auch den Patterngedanken verfolgen. Wie auch in Perl, Python und Java basieren die in Xcerpt verwendeten regulären Ausdrücke auf dem POSIX Standard[7]. Die meisten Entwickler sind wohl mit der Handhabung von regulären Ausdrücken vertraut. Aus diesem Grund wird an dieser Stelle auf eine detaillierte Einführung verzichtet.

Charakterklassen: Ein in Xcerpt-Programme häufig verwendetes Konstrukt der regulären Ausdrücke sind Charakterklasse. Anhand von Charakterklassen werden die Zeichen, mit denen ein regulärer Ausdruck matchen soll, eingeschränkt. Charakterklassen werden durch eckige Klammern, die eine Menge von Zeichen umschließen, definiert. Im POSIX Standard gibt es eine Anzahl von vordefinierten Charakterklassen, die durch reservierte Wörter, umschlossen von Doppelpunkten, beschrieben werden. So matcht die Charakterklasse `[:alnum:]` mit allen alphanumerischen Zeichen. Eine benutzerdefinierte Charakterklasse, die mit den Ziffern "1", "4" und "6" matcht, wird mit `[146]` definiert, eine Charakterklasse, die mit den Ziffern "1" bis "9" matcht mit `[1-9]`.

In Xcerpt können reguläre Ausdrücke nicht nur an der Stelle eines Literals stehen, sondern auch für den Bezeichner eines Terms, was Einschränkungen von Anfragen stark begünstigt. Wie in Perl, stehen reguläre Ausdrücke auch in Xcerpt immer innerhalb von zwei "Slashes".

Ein Anfrageterm der Gestalt:

```
title{/.*XML.*/}
```

matcht mit allen `title`-Termen, die den Substring "XML" beinhalten.

Ein Anfrageterm der Gestalt:

```
/.*XML.*/{{}}
```

matcht mit allen Bezeichnern, in denen der Substring "XML" vorkommt.

Erweiterungen von regulären Ausdrücken: Reguläre Ausdrücke nach dem POSIX Standard beinhalten nicht die volle Funktionalität wie sie in Xcerpt benötigt wird. Zwar unterstützen sie die Idee von Pattern, allerdings fehlt es noch an Möglichkeiten, Zeichenket-

[4] http://www.perl.com
[5] http://www.onlamp.com/python
[6] http://java.sun.com
[7] http://standards.ieee.org/regauth/posix

ten, die mit regulären Ausdrücken übereinstimmen über Bindungen an Variablen in Xcerpt-Programmen zu nutzen. Zu diesem Zweck sind die regulären Ausdrücke in Xcerpt um eingeschränkte Variablen ("`->`") erweitert. Jeden reguläre Ausdruck kann man mit in einer Untergruppe schreiben: Der Ausdruck `/.*/` ist äquivalent zu `/(.*)/`. Innerhalb von Untergruppen können die Variablendeklarationen erfolgen.

Der folgende Anfrageterm bindet den kompletten Inhalt eines `title`-Terms an die Variable `X`, falls in dem Inhalt der Substring "XML" auftritt.

```
title {/(var X -> .*XML.*)/}
```

2.8.7 Namensräume

Führt man verschiedene XML Dokumente zusammen, die unterschiedlichen Schemata zugrunde liegen aber gemeinsame Elementnamen beinhalten, kann es zu Namenskonflikten kommen. Um diese Mehrdeutigkeiten zu vermeiden benutzt man Namensräume (Namespaces). Jedes Element kann mit einem Namensraum versehen werden, welcher durch eine IRI definiert wird. In XML kann die Deklaration von Namensräumen innerhalb eines jeden Elements geschehen. Ein Namensraum wird deklariert, indem man der gewünschten IRI ein Präfix zuordnet. Unterschiedliche Präfixe können hierbei dem selben Namensraum zugeordnet werden. Ein Element wird mit einem Namensraum gekennzeichnet, indem vor dem Elementnamen das Präfix gefolgt von einem Doppelpunkt geschrieben wird.

Xcerpt unterstützt den Namensraummechanismus. Die Deklaration von Namensräumen kann aber zum jetzigen Entwicklungsstand von Xcerpt nur am Anfang eines Programms geschehen, also vor den Regeln und Zielen. Somit ist Verschachtelung und Überschattung von Namensräumen noch nicht möglich. Ein Namensraum wird wie folgt deklariert:

```
ns-prefix <identifier> = <iri>
```

Während in XML nur Präfixe vor den Elementnamen zugelassen sind, ist es in Xcerpt auch erlaubt, die IRI's vor den Elementnamen anzugeben. Anstelle der Präfixe können in Xcerpt-Anfragen auch Variablen zur Selektion und reguläre Ausdrücke zum Patternmatchen stehen. Hierbei werden allerdings nicht die Präfixe betrachtet, sondern die dahinterstehende IRI's. Ein regulärer Ausdruck wird also auf die IRI angewandt, eine Variable an die IRI gebunden. Variablen liefern somit auch die IRI's in Konstruktionstermen. Es bleibt zu klären, wie man in Konstruktionstermen optional auch das Präfix ausgeben kann, was in Transformationsanwendungen durchaus wünschenswert ist. In Abschnitt 3.1.7 findet man einige Anfragen über Namensräume.

2.8.8 Unifikation

Xcerpt wird als eine Anfragesprache gesehen, die der logischen Programmiersprache Prolog sehr ähnlich ist. Mit den bisher eingeführten Konstrukten ist eine Übertragung von Prologregeln in Xcerpt-Regeln aber noch nicht möglich.

Betrachte man das Prologprädikat

```
p([X|Y],Y).
```

Dieses Prädikat schneidet den Kopf des ersten Arguments, wenn es sich um eine Liste mit

mindestens einem Element handelt, ab und bindet die Restliste an die Variable `Y`. In Xcerpt kann man ein solches Prädikat nicht einfach angegeben werden, da dies eine Regel ohne Rumpf ist. Xcerpt-Regeln müssen sich aber aus einem Konstruktionsteil (Kopf) und wenigstens einem Anfrageteil (Rumpf) zusammensetzen. Eine andere, äquivalente Schreibweise für das Prologprädikat, welches Kopf und Rumpf besitzt, wäre die Regel

```
p(X,Y):- X = [Z|Y]
```

und kann folgendermaßen gelesen werden: Wenn das Eingabeargument `X` mit einer Liste, die sich aus einem Kopf `Z` und einem Rest `Y` zusammensetzt, vereinheitlicht (unifiziert) werden kann, dann wird `Y` zurückgegeben.

Eine Übersetzung in eine Xcerpt-Regel ist aber dennoch nicht möglich, da diese Vereinheitlichung von zwei Termen in dem Anfrageteil nicht ausdrückbar ist; die Idee von Anfragetermen ist, Datenterme mit einem Anfragepattern zu matchen. Um diese besondere Art von Regeln dennoch zuzulassen, wird ein Unifikationskonstrukt "`:<`" eingeführt. Die der Prologregel entsprechende Xcerpt-Regel, unter Berücksichtigung der nötigen Termstruktur und Sammelkonstrukte, lautet dann:

```
CONSTRUCT
p [
  var X,
  result [
    all var Y
  ]
]
FROM
list [[
  var Z,
  var Y
]]:< var X
END
```

Ein Aufruf dieser Regel kann über ein Ziel erfolgen.

```
GOAL
  var R
FROM
  p [list [ a, b, c ], var R]
END
```

In dem Anfrageteil des Ziels wird versucht den Term `p` zu erfüllen. Da dieser Term mit dem Konstruktionsteil der Regel matcht, wird versucht, deren Anfrageteil zu erfüllen. Dies gelingt, da der Anfrageterm mit der Variablen `X` unifiziert werden kann, da an ihr der Term `list[a, b, c]` gebunden ist. Die Variable `Z` kann entweder an `a{}` gebunden und `Y` dann folglich an `b{}` oder `c{}`, oder aber `Z` wird an `b{}` gebunden und `Y` kann nur an `c{}` gebunden. Im Konstruktionsteil werden innerhalb des `result`-Terms alle möglichen Bindungen für `Y` ausgegeben, also `b{}` und `c{}`. Der Anfrageteil des Ziels kann durch die Bindung der Variable `R` an den `result`-Term erfüllt werden. Der Ergebnisterm ist dann `result[b, c]`. Es ist zu beachten, dass erst das `all`-Konstrukt im Kopf der Regel auch den Term `c{}` liefert. Ohne `all` würde nur `b` im Ergebnis stehen.

Kapitel 3

Anwendungsszenarien (Use Cases)

Anhand von Anwendungsbeispielen lassen sich recht schön die Fähigkeiten einer Anfragesprache hervorheben. Durch verschiedenartige Dokumentstrukturen werden unterschiedliche Anforderungen an eine Anfragesprache gestellt. Das Ziel ist es, durch Implementierung von Anwendungsszenarien Stärken, Schwächen und Verbesserungsmöglichkeiten von Xcerpt herauszuarbeiten.

3.1 "XML Query Use Cases" vom W3C

Die "XML Query Working Group" des W3C erarbeitete einen Entwurf, in welchem Anwendungsszenarien [DCPFDF+03] für die XML Anfragesprache XQuery spezifiziert wurden. In diesem Dokument werden wichtige Anwendungen für eine XML Anfrage Sprache dargestellt, welche in unterschiedlichen Anwendungsszenarien aufgeteilt wurden und welche sich auf spezielle Anwendungsgebiete beziehen.

Diese Anwendungsszenarien wurden im Rahmen dieser Diplomarbeit in der Sprache Xcerpt (in ihrem momentanen Entwicklungszustand) implementiert, mit dem Ziel, konzeptuelle Vor- wie auch Nachteile der Sprache zu identifizieren und fehlende Konstrukte der Syntax und Funktionalitäten wie zum Beispiel noch nicht implementierte Operationen und Funktionen ausfindig zu machen. Allerdings wird in diesem Kapitel nur eine Auswahl von Anfragen vorgestellt und näher erläutert um dem Leser einerseits ein Gefühl für die Sprache anhand von einfacheren Anfragen zu vermitteln, andererseits mittels komplexer Beispiele weiterführende Möglichkeiten und Schwierigkeiten dieser patternbasierten, logischen Anfragesprache nahe zu bringen. Noch fehlende Konstrukte der Sprache werden als Pseudolösung in die Anfragen mit aufgenommen und gesondert hervorgehoben. Alle Anfragen, inklusive der hier nicht vorgestellten, sind im Anhang nochmals zu finden.

Die hier vorgestellten Anfragen erfüllen meistens folgende Gesichtspunkte:

- Anfragen, die wegen des positionellen Aspekts von Xcerpt gut zu lösen sind
- Anfragen, die besondere Funktionen benötigen (Aggregation, Arithmetik)
- Rekursive Anfragen

- Anfragen mit "Rule Chaining"
- Anfragen über mehrere Dokumente und Terme mit Wertevergleich ("Joins")

Die Gliederung der Use Cases des W3C's wird beibehalten um das Nachschlagen im Anhang 5 und auf der Web-Seite des W3C [DCPFDF+03] zu erleichtern.

Die Art der Klammern von Anfrage- und Konstruktionstermen wird nur in Beispielen angesprochen, in denen die Ordnung beachtet werden muss. Oft werden aber auch eckige Klammern verwendet, weil die Struktur der XML Dokumente hinlänglich bekannt ist, auch wenn die Reihenfolge des Auftretens von Elementen keine Rolle spielt.

3.1.1 Use Case: "XMP Experiences and Exemplars"

Die Anfragen dieses Anwendungsszenarios stützen sich auf mehrere XML Dokumente, die verschiedene Informationen zu Büchern beinhalten.

Die erste Datei "bib.xml" enthält eine Beispielliste von Büchern, wie sie intern für die Seite "www.bn.com" abgespeichert werden. Bücher werden mit den Elementen `title`, `author`, `publisher` und `price` beschrieben und einem Erscheinungsdatum des Buches als Attribut. Eine zweite Datei "reviews.xml" enthält Einträge über Bücher, die von "www.amazon.com" angeboten werden und durch die Elemente `title`, `price` und `review` beschrieben werden. Die Datei "books.xml" enthält eine Inhaltsangabe eines Kapitels eines unbekannten Buches. Dieses Kapitel enthält Abschnitte (`section`), die wiederum Unterabschnitte enthalten. Das Dokument "prices.xml" enthält Preisangaben von Büchern, abhängig von den Online-Shops wo sie angeboten werden. Alle Dokumente sind im Anhang in Abschnitt 5.1.1.1 zu finden.

3.1.1.1 Q1: Grundlegendes Beispiel

List books published by Addison-Wesley after 1991, including their year and title.

Um diese Anfrage zu schreiben müssen in dem Anfragepattern (Zeile 20-31) das Jahr und der Titel an Variablen gebunden werden. Hierbei ist zu beachten:

- Das Pattern wird von der Wurzel (`bib`) ab spezifiziert. Dies ist aber nicht unbedingt nötig. Eine ausreichende Selektion könnte man auch über das `descendant`-Konstrukt, geschrieben vor `book`, erreichen.
- Die ordnungserhaltenden Klammern sind nicht nötig, eine ungeordnete Spezifikation würde ausreichen. Allein bei der Selektion des Attributs `year` darf, bedingt durch die Definition von Attributen, keine geordnete Termspezifikation angegeben werden.
- Es ist ausreichend, die Variable `Title` an den `title`-Term zu binden und nicht an dessen Inhalt, während hingegen die Variable `Year` an den Inhalt gebunden werden muss, um einen Wertevergleich in der "Condition Box" (Zeile 32) durchführen zu können. Da Typüberprüfung und Typinferenz noch nicht in Xcerpt existieren, muss `Year` mit einem Typen versehen werden, damit der Vergleich in der "Condition Box" gelingt.
- Im Konstruktionsterm werden alle Titel und Jahresangaben innerhalb eines `book`-Terms wieder angeordnet. Während die Variable für den Titel direkt unter dem `book`-Term

erscheinen kann, muss die Variable für das Jahr explizit in einem `year`-Term unter einem `attributes`-Term angegeben werden. Nur so wird ein Datenterm erzeugt, der eine korrekte Attributsspezifikation enthält.

- Zur Ausgabe der Ergebnisse im HTML Format wird ein Ziel verwendet, in welchem die Wurzel `bib` des neu erzeugten Terms gebunden und wieder ausgegeben wird.

```
GOAL
out {
  resource { "stdout:", "html" },
  var C
}
FROM
var C -> bib {{}}
END

CONSTRUCT
bib [
  all book [
    attributes  {
      year [ var Year ]
    },
    var Title
  ]
]
FROM
in {
  resource [ "file:bib.xml", "xml" ],
  bib [[
    book [[
      attributes  {{
        year { var Year::integer }
      }},
      var Title -> title {{}},
      publisher { "Addison-Wesley" }
    ]]
  ]]
} where { var Year::integer gt 1991 }
END
```

3.1.1.2 Q5: Anfrage mit Join durch Konjunktion von Anfragetermen

For each book found at both xmp-bib.xml and xmp-reviews.xml, list the title of the book and its price from each source.

Diese Anfrage benötigt ein Pattern, welches nur Bücher mit dem selben Titel aus zwei verschiedenen Ressourcen zusammenführt. Dies erreicht man, indem man zwei Anfrageterme mit einem `and{}` verknüpft und somit verlangt, dass beide Anfrageterme erfüllt sein müssen. Die Bedingung, ein Buch gleichen Namens aus beiden Quellen zu erhalten, kann durch den gemeinsamen Variablennamen innerhalb des `title`-Terms erreicht werden und entspricht einem Equijoin. Die jeweiligen Preise werden durch die Variablen `Pa` und `Pb` selektiert und durch den Konstruktionsterm in der gewünschten Form ausgegeben.

```
CONSTRUCT
books-with-prices [
  all book-with-prices [
    title [
      var T
    ],
    price-amazon [ var Pa ],
    price-bn [ var Pb ]
  ]
]
FROM
and {
  in {
    resource { "file:xmp-bib.xml", "xml" },
    bib [[
      book [[
        title [ var T ],
        price [ var Pa ]
      ]]
    ]]
  },
  in {
    resource { "file:xmp-reviews.xml", "xml" },
    reviews [[
      entry [[
        title [ var T ],
        price [ var Pb ]
      ]]
    ]]
  }
}
END
```

3.1.1.3 Q6: Anfrage mit Fallunterscheidung

For each book that has at least one author, list the title and first two authors, and an empty “et-al” element if the book has additional authors.

Betrachte man zunächst die Anfrage in XQuery:

```
<bib>
  {
    for $b in doc("http://bstore1.example.com/bib.xml")//book
    where count($b/author) > 0
    return
        <book>
            { $b/title }
            {
                for $a in $b/author[position()<=2]
                return $a
            }
            {
                if (count($b/author) > 2)
                 then <et-al/>
                 else ()
            }
        </book>
  }
</bib>
```

Wie erwähnt ist in XQuery eine Vermischung von Selektion und Konstruktion möglich. So wird innerhalb eines Konstruktionsteils (eingeleitet durch das Schlüsselwort `return`) eine weitere Selektion vorgenommen (Zeile 9) und über deren Ergebnisse wieder neue Ergebnisse kreiert (Zeile 10). Wertevergleiche, Schleifen und bedingte Anweisungen sind überall verfügbar.

Dieselbe Anfrage wird in Xcerpt aus folgenden Gründen durch mehrere Regeln ausgedrückt:

- Xcerpt bietet die Aggregationsfunktion `count` an. Wie in Abschnitt 2.8.1.3 beschrieben, kann `count` nur in Konstruktionstermen verwendet werden, da diese Funktion nur auf die Menge der im Anfrageteil gefundenen Variablenbelegungen angewandt werden kann. Dies bedingt allerdings, dass das Ursprungsdokument in einer Vorbereitungsregel um die Information der Autorenanzahl angereichert werden muss. Nur so kann man dann in weiteren Regeln die gewünschten unterschiedlichen Ergebnisterme erzeugen, die von der Autorenanzahl abhängig sind.

- Um die Anzahl der Autoren herauszufinden, werden anhand einer Regel (Zeile 46-64) alle Bücher und deren Autoren an Variablen gebunden und im Konstruktionsterm innerhalb eines neuen `books`-Terms angeordnet. Zu beachten ist auch, dass die Anzahl der Autoren nicht einfach innerhalb des `book`-Terms eingefügt werden kann, der an die Variable `Book` gebunden ist (Zeile 49). Dies wäre nur möglich, wenn alle Subterme von `book` in der Anfrage an Variablen gebunden werden und `book` explizit mit allen alten und neuen Elementen wieder konstruiert werden würde.

- Weiterhin ist zu beachten, dass diese Regel nur Bücher mit deren Anzahl an Autoren liefert, wenn ein Buch mindestens einen Autor hat. Ansonsten kann das Pattern nicht auf den Datenterm angewandt werden und scheitert, was aber in diesem Fall der Spezifikation der Anfrage entspricht. Würde man auch Bücher ohne Autoren selektieren wollen, könnte man ein `optional` vor der Variablendeklaration (Zeile 60) angeben. `optional` müsste dann auch in Zeile 51 nach dem `all` angegeben werden; `count` liefert dann auch den Wert "0", wenn es keine Autoren gibt.

- Die zweite Regel (Zeile 28-44) wird erfüllt, wenn es in dem neuen Datenterm mit Wurzel `books-with-authorcount` ein Buch mit Titel, Autoren und dem neuen Term `authorcount` gibt, wobei die Anzahl der Autoren zwischen "1" oder "2" betragen muss. Diese Bedingung wird in der "Condition Box" in Zeile 43 ausgedrückt.

- Die erste Regel (Zeile 9-26) gibt alle Bücher mit Titel, die ersten beiden Autoren und einem leeren `et_al` Element zurück, wenn es mehr als zwei Autoren gibt. Da die erste und zweite Regel sich gegenseitig wegen der Bedingung in der "Condition Box" ausschließen, gibt es keine unerwünschten Mehrfachergebnisse. In der ersten und zweiten Regel werden die Unterelemente von `book` über ihrer Ordnung befragt, damit die Reihenfolge, in der die Autoren auftreten, erhalten bleibt und auch wirklich die ersten beiden Autoren im Konstruktionsteil zurückgegeben werden können.

- Um die ersten beiden Autoren auszugeben, wird `first 2` in der ersten Regel verwendet (Zeile 12).

```
GOAL
  result { all var }
FROM
  var C -> book {{}}
END

CONSTRUCT
  book {
    var Title,
    first 2 var Author,
    et_al []
  }
FROM
  books_with_authorcount {{
    books {{
      book {{
        var Title -> title {{}},
        var Author -> author {{}}
      }},
      authorcount { var Count::integer }
    }}
  }}
  where { var Count::integer geq 3 }
END

CONSTRUCT
  book {
    var Title,
    all var Author
  }
FROM
  books_with_authorcount {{
    books {{
      book {{
        var Title -> title {{}},
        var Author -> author {{}}
      }},
      authorcount { var Count::integer }
    }}
  }}
  where { var Count::integer leq 2 }
END

CONSTRUCT
  books_with_authorcount [
    all books [
      var Book,
        authorcount {
          count ( all var Author )
        }
      ]
  ]
FROM
  in {
    resource { "file:xmp-bib.xml", "xml" },
    bib [[
      var Book -> book {{
          var Author -> author {{}}
        }}
    ]]
  }
END
```

3.1.1.4 Q8: Reguläre Ausdrücke und Suche in beliebiger Tiefe

Find books in which the name of some element ends with the string "or" and the same element contains the string "Suciu" somewhere in its content. For each such book, return the title and the qualifying element.

Diese Anfrage enthält zwei Besonderheiten. Zum einen werden zwei reguläre Ausdrücke verwendet. Der eine muss mit einem Bezeichner matchen, der andere mit dem Inhalt eines Terms. In Zeile 14 der folgenden Xcerpt-Regel werden alle Terme an die Variable `X` gebunden, deren Bezeichner mit "or" enden. Mit dem zweiten regulären Ausdruck wird der Inhalt dieser Terme nach dem Substring "Suciu" durchsucht. Um aber alle Subterme unabhängig von der Schachtelungstiefe nach diesem Substring zu durchsuchen, wird `descendant` verwendet.

```
1  CONSTRUCT
2  bib [
3    all book [
4      var Title,
5      var X
6    ]
7  ]
8  FROM
9  in {
10   resource { "file:xmp-bib.xml", "xml" },
11   bib [[
12     book {{
13       var Title -> title {{}},
14       desc var X -> /.*or/ {{
15         desc /.*Suciu.*/
16       }}
17     }}
18   ]]
19 }
20 END
```

3.1.1.5 Q9: Disjunktion über Anfrageterme

In the document "books.xml", find all section or chapter titles that contain the word "XML", regardless of the level of nesting.

Der `or{}` Term verknüpft zwei Anfrageterme über ein nicht exklusives "or". Xcerpt versucht also beide Terme zu erfüllen. In diesem Beispiel werden im Konstruktionsteil alle Bindungen beider Anfrageterme ausgegeben.

```
1  CONSTRUCT
2  results [
3    all var Title
4  ]
5  FROM
6  in {
7    resource { "file:xmp-books.xml", "xml" },
8    or {
9      desc chapter [[
10       var Title -> title [[ /.*XML.*/ ]]
11     ]],
```

```
12      desc section [[
13        var Title -> title [[ /.*XML.*/ ]]
14      ]]
15    }
16  }
17  END
```

3.1.1.6 Q12: Mächtigkeit von ungeordneten Anfragen

Find pairs of books that have different titles but the same set of authors (possibly in a different order).

Diese scheinbar einfache und kurze Anfrage hat es in sich; zumindest, wenn man sie mit XQuery lösen möchte. Oder anders gesagt: Für diese Art von Anfragen ist Xcerpt geschaffen! Man betrachte zuerst das entsprechende XQuery Programm.

```
1   <bib>
2   {
3       for $book1 in doc("http://bstore1.example.com/bib.xml")//book,
4           $book2 in doc("http://bstore1.example.com/bib.xml")//book
5       let $aut1 := for $a in $book1/author
6                    order by $a/last, $a/first
7                    return $a
8       let $aut2 := for $a in $book2/author
9                    order by $a/last, $a/first
10                   return $a
11      where $book1 << $book2
12      and not($book1/title = $book2/title)
13      and deep-equal($aut1, $aut2)
14      return
15          <book-pair>
16              { $book1/title }
17              { $book2/title }
18          </book-pair>
19  }
20  </bib>
```

Was dieses XQuery Programm so komplex macht, ist die Berücksichtigung der Ordnung. Die Autoren müssen vorerst sortiert werden, damit man sie überhaupt miteinander vergleichen kann.

Mit Xcerpt ist diese Anfrage durch (un-)geordnete Spezifikation intuitiver zu lösen. Es werden zwei Regeln benötigt.

- **Regel von Zeile 20-37:** Alle Autoren sind in der Datenbank direkte Kindterme von `book`. Will man die Menge der Autoren vergleichen, muss man sie in einem eigenen Term anordnen. Es werden daher alle Autoren eines Buches aufgesammelt, und innerhalb eines neuen Terms `author_set` eingefügt.
- **Regel von Zeile 1-18:** In dieser Regel vergleicht man immer ein Tupel von `title_authorset`-Termen, ob diese unterschiedliche Titel (ausgedrückt durch die unterschiedlichen Variablen `Title1` und `Title2`) aber `author_set`-Terme mit gleichem Inhalt besitzen (gemeinsame Variable `author_set`). Im Gegensatz zu der XQuery Anfrage muss man nicht sortieren. Die ungeordnete Spezifikation (`author_set`) in Zeile 12

und 16 ist ausreichend. Der Unifikationsalgorithmus von Xcerpt übernimmt die Aufgabe, diese beiden Terme auf Gleichheit zu überprüfen. Wichtig ist allerdings das Beachten der Ordnung über die `title_authorset`-Terme (Zeile 9 und 18). Dadurch werden die beiden Terme in ihrer Reihenfolge betrachtet ("der zweite Term kommt nach dem ersten Term"). Dies ist nötig, damit ein Term nicht mit sich selber verglichen wird, wie es bei einer ungeordneten Spezifikation der Fall wäre. Im Ergebnisterm `titlebib` würden ansonsten auch die Tupel mit gleichen Titeln stehen.

```
1  CONSTRUCT
2  bib {
3    all titlebib {
4      title { var Title1 },
5      title { var Title2 }
6    }
7  }
8  FROM
9  temp [[
10   title_authorset {{
11     title { var Title1},
12     var Author_set -> author_set {{}}
13   }},
14   title_authorset {{
15     title { var Title2},
16     var Author_set -> author_set {{}}
17   }}
18 ]] where { var Title1 ne Title2 }
19
20 CONSTRUCT
21 temp {
22   all title_authorset {
23     title { var Title },
24     author_set { all var Author }
25   }
26 }
27 FROM
28 in {
29   resource { "file:xmp-bib.xml", "xml" },
30   bib [[
31     book {{
32       title { var Title },
33       var Author -> author {{}}
34     }}
35   ]]
36 }
37 END
```

3.1.2 Use Case "TREE: Queries That Preserve Hierarchy"

XML Anfragesprachen müssen in der Lage sein unter Beibehaltung der Struktur Elemente zu extrahieren. Dieser Anwendungsfall beinhaltet solche strukturellen Anfragen und verwendet hierbei ein Dokument (zu finden in Abschnitt 5.1.2.1, welches auf einer DTD beruht, die ein hohes Maß an Heterogenität zulässt.

3.1.2.1 Q1: Rekursive Regeln

Prepare a (nested) table of contents for Book1, listing all the sections and their titles. Preserve the original attributes of each ¡section¿ element, if any.

Will man einen Baum nach Elementen, dessen Tiefe nicht bekannt ist, durchsuchen und aus diesen Elementen wieder einen Baum aufbauen so erscheint eine rekursive Anfrage als geeignet. Das folgende XQuery Programm benutzt hierzu eine Funktion, die sich selbst wieder rekursiv aufruft.

```
1  declare function local:toc($book-or-section as element()) as element()*
2  {
3      for $section in $book-or-section/section
4      return
5        <section>
6           { $section/@* , $section/title , local:toc($section) }
7        </section>
8  };
9
10 <toc>
11    {
12      for $s in doc("book.xml")/book return local:toc($s)
13    }
14 </toc>
```

Xcerpt ist eine logikbasierte Anfragesprache die stellenweise mit Prolog vergleichbar ist. Daher werden rekursive Regeln zuerst anhand eines Prolog Beispiels motiviert und danach das Xcerpt-Programm vorgestellt.

Die Aufgabe der Query könnte man vereinfacht folgendermaßen beschreiben: “Das Ergebnis einer Transformation eines Baumes ist wiederum ein Baum, der aus bestimmten Knoten und Blätter des Eingabebaums besteht”. Intuitiv sollte klar sein, dass man den kompletten Eingabebaum durchgehen muss und an jedem Knoten entscheiden muss, ob dieser für die Ausgabe relevant ist oder nicht.

Um dieses Problem zu vereinfachen, reduzieren wir den Baum zu einer Liste und versuchen aus einer Liste als Eingabe, eine andere Liste zu rekonstruieren. Diese Problem könnte in Prolog folgendermaßen dargestellt werden (siehe [LSAS99]).

```
transform ([ ], [ ]).
transform ([ Head | Tail ], [ NewHead | NewTail ]:-
        change ( Head, NewHead ) , transform ( Tail, NewTail).
```

Hierbei besagt das erste `transform`-Prädikat, dass eine leere Liste wiederum die leere Liste ist, was man auch als Basisfall bezeichnet. Die Hauptaufgabe des Prädikats `transform` besteht aber darin, den Kopf der Eingabeliste eventuell umzuwandeln und ihn als Kopf in der Ausgabeliste zu verwenden, dann den Rest der Eingabeliste mit `transform` und den Rest der Ausgabeliste für den geänderten Rest stehen zu lassen. Wenn das Ende der Eingabeliste erreicht ist, gibt es kein Listenelement mehr, welches in die Ausgabeliste aufgenommen werden müsste, so dass die Ausgabeliste mit einer leeren Liste abgeschlossen werden kann. Eine Formulierung, die deklarativ wäre und somit Prolog mehr entspricht, wäre: “Das Transformieren einer Liste, welche aus einem Kopf `Head` und einem Rest `Tail` zusammengesetzt ist, ergibt

eine Liste mit dem Kopf `NewHead` und dem Rest `NewTail`, wenn sich durch Umwandeln des Listenelements `Head` das Element `NewHead` und durch Transformieren der Liste `Tail` die Liste `NewTail` ergibt." Das Prädikat `change` wird hier nicht näher beleuchtet, soll aber sowohl ein neues Element erzeugen, wie auch das alte übernehmen können.

In ähnlicher Weise müsste man die Anfrage in Xcerpt schreiben, nur dass man nun nicht mehr eine Liste betrachtet, sondern Terme. Angenommen, man will den Eingabeterm nicht verändern, sondern nur komplett durchlaufen: "Der Durchlauf eines Terms mit der Wurzel `Label` und deren Subtermen `SubTerm` ergibt eine Menge von Termen `NewSubterm`, wenn sich durch ein Durchlaufen aller Subterme `SubTerm` die Menge der Terme `NewSubTerm` ergibt."

Das komplette Xcerpt-Programm hat folgende Gestalt, wobei zu beachten ist, dass als Eingabeterm immer eine Variable verwendet wird. Im Anfrageteil wird versucht, diese mit einem Pattern zu unifizieren, was über den Unifikationsoperator "`:<`" geschieht.

```
GOAL
var Result
FROM
and {
  in {
    resource { "file:book-tree.xml", "xml" },
    var Root -> book {{}}
  },
  transform [ var Root, var Result ]
}
END

CONSTRUCT
  transform [
    var In ,
    section [
      optional var Attributes
      titel { var Titel },
      all var NewSubterm
    ]
  ]
FROM
  and {
    section {{
      optional var Attributes -> attributes {{}}
      title { var Titel },
      desc var Rest -> section {{}}
    }} :< var In,
    transform [ var Rest, var NewSubterm ]
  }
END

CONSTRUCT
  transform [
    var In,
    section [
      optional var Attributes
      titel { var Titel }
    ]
  ]
FROM
  section {{
    optional var Attributes -> attributes {{}}
    title { var Titel },
```

```
45      without desc var Rest -> section {{}}
46    }} :< var In
47  END
48
49  CONSTRUCT
50    transform [
51      var In,
52      result [ all var NewSubterm ]
53    ]
54  FROM
55    and {
56      var Label {{
57        var Subterm
58      }} where { var Label != "section" }
59      :< var In,
60      transform [ var Subterm, var NewSubterm ]
61    }
62  END
```

Das Xcerpt-Programm besteht aus drei `transform`-Regeln die sich gegenseitig ausschließen. An jedem Knoten (in Xcerpt, Bezeichner eines Terms) wird entschieden, welche der Regeln auf den Term angewandt wird. Die erste Regel schaltet nur dann, wenn der Eingabe Term, gebunden an die Variable `In`, mit einem Term unifiziert werden kann, der eventuell Attribute, einen `title`-Term und weitere `section`-Terme in beliebiger Tiefe besitzt.

Die zweite Regel ist eine Terminierungsregel. Sie wird nur angewandt, wenn die Eingabe mit einem Term unifiziert werden kann, der zwar optionale Attribute und einen Titel besitzt, aber keine weiteren `section`-Terme in ihm enthalten sind.

Die dritte Regel schaltet nur, wenn es sich bei dem Eingabeterm nicht um einen `section`-Term handelt, wird also zur "Überbrückung" von der Wurzel bis zum ersten Auftreten eines `section`-Elementes verwendet. Zur Lösung der Anfrage ist sie nicht unbedingt nötig, da man die `Root`-Variable des Zieles in Zeile 7 direkt an die ersten Kapitel binden könnte. Allerdings ist diese Regel eine sehr allgemeine. Ohne der "Condition Box" würde diese Regel den kompletten Baum durchlaufen ohne eine Ausgabe zu erzeugen. Sie bietet sich daher für jegliche Rekursion über Bäume an.

3.1.2.2 Q3: Aggregation in Verbindung mit `optional` I

How many sections are in Book1, and how many figures?

In dieser Anfrage ist darauf zu achten, dass ein `optional` vor den Variablen verwendet wird. Würde es nämlich kein `section`- oder `figure`-Element geben, würde `count` im Konstrukionsteil kein Ergebnis liefern. Fatal wäre dies vor allem, wenn weder `section`- noch `figure`-Terme in der Datenbank stehen; die Anfrage würde scheitern. Mit einem `optional` hingegen kann `count` den Wert "0" liefern.

```
1  CONSTRUCT
2  count [
3    section_count {
4      count ( all optional var Section )
5    },
6    figure_count {
```

```
7       count ( all optional var Figure )
8     }
9   ]
10  FROM
11  in {
12    resource { "file:book-tree.xml", "xml" },
13    and {
14      desc optional var Section -> section {{}},
15      desc optional var Figure -> figure {{}}
16      }
17    }
18  END
```

3.1.3 Use Case "SEQ - Queries Based on Sequences"

In Anfragen auf Sequenzen muss prinzipiell die Reihenfolge der Terme über geordnete Termspezifikationen beachtet werden. Als Datenbank (in Abschnitt 5.1.3.1) wird ein medizinischer Bericht verwendet, dessen Inhalt durch Aktionen bestimmt ist, die zeitlich nacheinander durchgeführt wurden. Häufig müssen Terme unter Berücksichtigung ihrer Position selektiert werden. Wegen der Trennung von Selektion und Konstruktion benötigt man hierzu immer mehrere Regeln.

3.1.3.1 Q1: Aggregation und Ordnung zur Positionsselektion

In the Procedure section of Report1, what Instruments were used in the second Incision?

Die zweite Regel (Zeile 11-23) sammelt alle `incision`-Terme, die innerhalb des Kapitels "Procedure" auftreten, unter Berücksichtigung der Ordnung auf und liefert die zweite `incision`, indem der erste Term mit `drop1` abgeschnitten und von der Restsequenz der erste ausgegeben wird. Die erste Regel muss lediglich alle Instrumente in dem Zwischenergebnis finden.

```
1   CONSTRUCT
2   result {
3     all var Instrument
4   }
5   FROM
6   temp {{
7     desc var Instrument -> instrument {{}}
8   }}
9   END
10
11  CONSTRUCT
12  temp [
13    first ( drop 1 ( all var Incision ) )
14  ]
15  FROM
16  in {
17    resource { "file:report1.xml", "xml" },
18    desc section [[
19      section.title { "Procedure" },
20      desc var Incision -> incision {{}}
21    ]]
22  }
23  END
```

3.1.3.2 Q3: Komplexe positionelle Anfrage I

In Report1, what Instruments were used in the first two Actions after the second Incision?

Die Selektion von Aktionen, die nach dem zweiten “Einschnitt” auftreten, kann man über ein Pattern erreichen, indem die zwei `incision`-Terme explizit angegeben werden (Zeile 21 und 22). Das Pattern beschreibt also einen Term mit unbekannten Bezeichnern, das alle Aktionen binden soll, die nach **mindestens** zwei Einschnitten folgen. Das **“mindestens”** ist zu beachten, da durch dieses Pattern nicht klar wird, ob es nicht noch mehr Einschnitte vor der Aktion gibt oder nicht sogar Aktionen vor oder zwischen den beiden Einschnitten stattgefunden haben. Entscheidend dabei ist aber nur, dass, bedingt durch die geordnete Spezifikation, die ersten beiden Bindungen der Variable `Action` die gesuchten sind und auch ausgegeben werden.

```
1  CONSTRUCT
2  result {
3    all var Instrument
4  }
5  FROM
6  temp {{
7    action {{
8      desc var Instrument -> instrument {{}}
9    }}
10 }}
11 END
12
13 CONSTRUCT
14 temp [
15   first 2 ( all var Action )
16 ]
17 FROM
18 in {
19   resource { "file:report1.xml", "xml" },
20   desc var X [[
21     incision {{}},
22     incision {{}},
23     desc var Action -> action {{}}
24   ]]
25 }
26 END
```

Wenn man allerdings Aktionen nach dem zwölften Einschnitt finden möchte, wird die Patternangabe sehr umständlich. Aus diesem Grund bietet es sich an, in einem Zwischenschritt die Daten so zu transformieren, dass sie über das `position`-Konstrukt befragt werden können.

```
1  CONSTRUCT
2  result {
3    all var Instrument
4  }
5  FROM
6  temp2 {{
7    action {{
8      desc var Instrument -> instrument {{}}
9    }}
10 }}
```

```
11 END
12
13 CONSTRUCT
14 temp2 {
15   first 2 ( action {all var Action})
16 }
17 FROM
18 temp1 [
19   position 2 I_A [[
20     var Action -> actions {{}}
21   ]]
22 ]
23
24 CONSTRUCT
25 temp1 [
26   all I_A [
27     var Inc,
28     all actions { optional var Action }
29   ]
30 FROM
31 in {
32   resource { "file:report1.xml", "xml" },
33   desc var X [[
34     var Inc -> incision {{}},
35     optional desc var Action -> action {{}}
36   ]]
37 }
38 END
```

Es werden zuerst alle Einschnitte einzeln selektiert und, wenn vorhanden, mit allen Aktionen, die darauf folgen in einem `I-A`-Term ausgegeben. Nun genügt es, den zweiten `I-A`-Term zu betrachten (Zeile 19), all seine Aktionen zu binden und die zweite davon auszugeben. Die letzte Regel liefert dann noch die gewünschten Instrumente.

3.1.3.3 Q4: Komplexe positionelle Anfrage II

In Report1, find "Procedure" sections where no Anesthesia element occurs before the first Incision.

Die Ausformulierung der Bedingung "es darf kein `anesthesia`-Element vor der ersten `incision` vorkommen" liest sich in XQuery fast wie in einer natürliche Sprache (Zeile 2-3).

XQuery Program

```
1 for $p in doc("report1.xml")//section[section.title = "Procedure"]
2 where not(some $a in $p//anesthesia satisfies
3         $a << ($p//incision)[1] )
4 return $p
```

In Xcerpt gibt es diese Beschreibungsmöglichkeit nicht. Stattdessen wird ein ähnlicher Ansatz wie in der vorherigen Anfrage verwendet.

Zunächst werden alle Einschnitte gebunden mit allen Termen, die vor diesem Einschnitt auftreten (Zeile 35 und 36). Zu beachten ist allerdings, dass als Ausgabe ein `section`-Term

verlangt wird. Dieser muss also “mitgeschleift” werden, indem er komplett an die Variable `Section` gebunden wird. Dieses Kapitel wird im Konstrukionsteil zusammen mit einem Term ausgegeben, der den ersten `incision`-Term mit all seinen Vorgänger-Termen beinhaltet. Die zweite Regel selektiert dann das Kapitel in dem `section_and_first_Inc`-Term (Zeile 7-13), in dem keine Narkose vor dem Einschnitt auftritt. Das Dokument “report1.xml” enthält beruhigender Weise keine Einschnitte, ohne dass vorher eine Narkose verabreicht wurde und die Xcerpt-Regel scheitert daher. Es wird also kein Ergebnis geliefert.

```
1  CONSTRUCT
2    result {
3      all var Section
4   }
5  FROM
6    temp [
7      section_and_first_Inc [[
8        var Section -> section {{}},
9        content [[
10         without anesthesia {{}},
11         incision {{}}
12       ]]
13     ]]
14   ]
15 END
16
17 CONSTRUCT
18   temp [
19     all section_and_first_Inc [
20       var Section,
21       first (
22         all content [
23           all optional var X,
24           var Inc
25         ]
26       )
27     ]
28   ]
29 FROM
30   in {
31     resource { "file:report1.xml", "xml" },
32     desc var Section -> section {{
33       section.title { "Procedure" },
34       section.content [[
35         optional var X,
36         var Inc -> incision {{}}
37       ]]
38     }}
39   }
40 END
```

3.1.3.4 Q5: Übersichtlichkeit durch Pattern

In Report1, what happened between the first Incision and the second Incision?

Dadurch dass Pattern ein “Ausschnitt” eines Datenterms darstellen, bieten sie ein hohes Maß an Lesbarkeit. Betrachte man die die entsprechende Anfrage als XQuery Programm.

```
declare function local:precedes($a as node(), $b as node()) as xs:boolean
{
    $a << $b
      and
    empty($a//node() intersect $b)
};


declare function local:follows($a as node(), $b as node()) as xs:boolean
{
    $a >> $b
      and
    empty($b//node() intersect $a)
};

<critical_sequence>
 {
  let $proc := doc("report1.xml")//section[section.title="Procedure"][1]
  for $n in $proc//node()
  where local:follows($n, ($proc//incision)[1])
    and local:precedes($n, ($proc//incision)[2])
  return $n
 }
</critical_sequence>
```

Dieses Programm beschreibt einen Algorithmus einer prozeduralen Programmiersprache. Durch die vielen verwendeten Funktionen fällt es nicht unbedingt leicht, den Algorithmus auf Anhieb zu verstehen.

Mit einer logikbasierten Sprache wie Xcerpt muss man sich nicht soviel Gedanken machen, wie man das Ergebnis erhält; man deklariert viel mehr, was man will. Diese Anfrage kann mit einer einzigen Regel verwirklicht werden, da alle Terme zwischen dem ersten und zweiten Einschnitt gesucht werden. Würde man Terme, die in einer Sequenz sehr spät auftreten, suchen, müsste man wieder mit mehreren Regeln und dem `posistion`-Konstrukt arbeiten. Entscheidend ist die Deklaration der Bezeichnervariable `Y`, die in der "Condition-box" genauer spezifiziert wird (`Y` und `Z` dürfen keine `incision` sein). Nur so kann gewährleistet werden, dass alle Terme, die an die Variable `Action` gebunden werden auch wirklich zwischen der ersten **und** zweiten `incision` liegen.

```
CONSTRUCT
result [
  all var Action
]
FROM
in {
  resource { "file:report1.xml", "xml" },
  desc var X [[
    optional var Y {{}},
    incision {{}},
    var Action,
    incision {{}}
  ]] where { var Y != "incision" }
}
END
```

3.1.4 Use Case “R - Access to Relational Data”

Relationale Datenbanksysteme sind die wohl am weitesten verbreiteten Datenbanksysteme überhaupt. Eine einfache Art, relationale Datenbanken nach XML zu übertragen gelingt, indem man für jede Tabelle (Relation) ein separates XML Dokument anlegt. Das Wurzelelement trägt den Namen der Tabelle, deren Unterelemente für die Tupel einer Tabelle stehen. Innerhalb der Tupel sind die Spalten der Tabelle angeordnet. Spaltenelemente, die Nullwerte zulassen sind in der DTD als optional gekennzeichnet. Ein fehlendes Element im Dokument bezeichnet dann einen Nullwert. Dieser Anwendungsfall beschreibt eine Auktion in der Benutzer (“users.xml”) Waren (“items.xml”) anbieten und Gebote abgeben (“bids.xml”). Anfragen auf diese drei Dokumente benötigen vor allem die Verwendung von Joins. Die Beispieldokumente liegen in Abschnitt 5.1.4.1 im Anhang.

3.1.4.1 Q1: Verwendung von Funktionen über den Typ `date`

List the item number and description of all bicycles that currently have an auction in progress, ordered by item number.

XQuery stellt eine Vielzahl von Funktionen bereit, die auch in Xcerpt erwünscht aber noch nicht implementiert sind. XQuery bietet für die Lösung dieser Anfrage die Funktion `current-date()` an. Das folgende Xcerpt-Programm verwendet dieselbe Funktion, mit der Annahme, dass sie schon existiert. Es ist zu beachten, dass Variablen, die an ein Datum gebunden werden sollen, mit dem Typ `date` an jeder Stelle versehen werden müssen.

```
CONSTRUCT
result {
  all item_tupel {
    itemno {
      var Itemno
    },
    var Descr
  } ordering [Itemno] numeric
}
FROM
in {
  resource { "file:items.xml", "xml" },
  items {{
    item_tuple {{
      itemno {{ var Itemno }},
      var Descr -> description {{
        /.*Bicycle.*/
      }},
      start_date { var Startdate::date },
      end_date { var Enddate::date }
    }}
  }}
}
where {
  and {
    var Startdate::date lt current-date(),
    var Enddate::date gt current-date()
  }
}
END
```

3.1.4.2 Q2: Aggregation in Verbindung mit `optional` II

For all bicycles, list the item number, description, and highest bid (if any), ordered by item number.

Wenn es eine Ware gibt, auf die geboten wurde, so gibt es auch einen entsprechenden Eintrag mit `itemno` und `bid` in der Datei "bids.xml". Gibt es also ein oder mehrere Gebote auf eine Ware, wird mit der Funktion `max` das höchste Gebot ermittelt und innerhalb eines `bid`-Terms ausgegeben (Zeile 8-10). Gibt es allerdings kein Gebot, dann soll auch kein `bid`-Term erzeugt werden. Daher wird im Anfrageterm ein `optional` vor dem Term `bid_tuple` verwendet (Zeile 27), um die Beziehung zwischen `itemno` und `bid` zu verdeutlichen. Denn es gibt nur ein Gebot, wenn es auch eine Warennummer gibt.

Analog schreibt man im Konstruktionsteil das `optional` vor dem `bid`-Term. Der Xcerpt-Interpreter erkennt alle mit `optional` gekennzeichneten Variablen (in diesem Fall die einzige Variable `Bid`) und erzeugt den gewünschten `bid`-Term mit dem höchsten Gebot, falls es eine Bindung für die Variable `Bid` gibt. Wenn nicht, wird auch kein `bid`-Term erzeugt.

```
1  CONSTRUCT
2  result {
3    all item_tupel {
4      itemno {
5        var Itemno
6      },
7      var Description,
8      optional bid {
9        max ( all var Bid )
10     }
11   } ordering [Itemno] numeric
12 }
13 FROM
14 and {
15   in {
16     resource { "file:items.xml", "xml" },
17     items {{
18       item_tuple {{
19         itemno {{ var Itemno }},
20         var Description -> description {{ /.*Bicycle.*/ }}
21       }}
22     }}
23   },
24   in {
25     resource { "file:bids.xml", "xml" },
26     bids {{
27       optional bid_tuple {{
28         itemno {{ var Itemno }},
29          bid { var Bid }
30       }}
31     }}
32   }
33 }
34 END
```

3.1.4.3 Q4: “Query Negation” mit `not`

List item numbers and descriptions of items that have no bids.

```
CONSTRUCT
result {
  all no_bid_item {
    itemno { var Itemno },
    description { var Description }
  }
}
FROM
and {
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        itemno { var Itemno },
        description { var Description }
      }}
    }}
  },
  not in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        itemno { var Itemno }
      }}
    }}
  }
}
END
```

Diese Xcerpt-Regel stützt sich auf folgende Eigenschaft des Dokuments: Die Regel liefert nur dann einen `itemno`- und `descriptions`-Term eines `item`-Terms aus der Datei “items.xml”, wenn dieses Item in der Datei “bids.xml” nicht eingetragen ist.

3.1.4.4 Q5: Schlechte Überschaubarkeit bei vielen Joins

For bicycle(s) offered by Tom Jones that have received a bid, list the item number, description, highest bid, and name of the highest bidder, ordered by item number.

Diese Anfrage erfordert mehrere Joins über die drei Dokumente “users.xml”, “items.xml” und “bid.xml”. In der XQuery Lösung sind diese Joins alle nach dem `where`-Ausdruck aufgelistet. Die gesamte Anfrage ähnelt in ihrer Gestalt einem “SQL Statement”, nur dass die Ausgabe in dem `return`-Teil noch verfeinert werden kann.

```
<result>
  {
    for $seller in doc("users.xml")//user_tuple,
        $buyer in  doc("users.xml")//user_tuple,
        $item in  doc("items.xml")//item_tuple,
        $highbid in  doc("bids.xml")//bid_tuple
    where $seller/name = "Tom Jones"
      and $seller/userid  = $item/offered_by
```

```
9         and contains($item/description , "Bicycle")
10        and $item/itemno  = $highbid/itemno
11        and $highbid/userid  = $buyer/userid
12        and $highbid/bid = max(
13                                  doc("bids.xml")//bid_tuple
14                                    [itemno = $item/itemno]/bid
15                              )
16      order by ($item/itemno)
17      return
18          <jones_bike>
19              { $item/itemno }
20              { $item/description }
21              <high_bid>{ $highbid/bid }</high_bid>
22              <high_bidder>{ $buyer/name }</high_bidder>
23          </jones_bike>
24    }
25  </result>
```

Das entsprechende Xcerpt-Programm ist zwar nicht komplexer, allerdings etwas schwerer zu lesen.

- **Regel von Zeile 46-100:** Diese Regel beinhaltet einen Join über aller drei XML Dokumente. So wird aus "user.xml" der Name des Benutzers genommen, der als Anbieter auftritt, dessen `iserid` also in "items.xml" unter `offered-by` gefunden wird. Dort wird auch die Beschreibung selektiert. Für dieses Item wird überprüft, welche Gebote alle abgegeben wurden (Join über `itemno`) und für diese Gebote wiederum wird wieder in "User.xml" nachgeschaut, welcher Benutzer dieses Gebot abgegeben hat (Join über `userid`). Als Zwischenergebnis werden für alle diese Items (welche Fahrräder sind) `itemno`, `description` und alle `user`, mit ihrem höchsten Gebot auf dieses Item, ausgegeben.
- **Regel von Zeile 30-44:** Es wird das höchste Gebot ermittelt, welches in dem Zwischenergebnis der ersten Regel zu finden ist.
- **Regel von Zeile 1-28:** Mit einem Join über die Ergebnisse der zweiten Regel (also dem höchsten Gebot) und ersten Regel wird das gewünschte Item mit dem höchsten Gebot ausgegeben.

Die zweite und dritte Regel könnte man auch durch eine Regel ausdrücken. Das Programm wird dadurch aber nicht übersichtlicher.

```
1   CONSTRUCT
2   result {
3     all jones_bike {
4       var Itemno,
5       var Description,
6       bid {
7         high_bidder { var Highbidder },
8         high_bid { var Highbid }
9       }
10    }
11  }
12  FROM
13  and {
```

```
  temp1 {{
    jones_bike {{
      var Itemno -> itemno {{}},
      var Description -> description {{}},
      bidandbidder {{
        high_bidder { var Highbidder },
        bid { var Highbid }
      }}
    }}
  }} ,
  temp2 {{
    maxbid { var Highbid }
  }}
}
END

CONSTRUCT
temp2 {
  maxbid {
    max ( all var Maxbid )
  }
}
FROM
temp1 {{
  jones_bike {{
    bidandbidder {{
      bid { var Maxbid }
    }}
  }}
}}
END

CONSTRUCT
temp1 {
  all jones_bike {
    itemno { var Itemno },
    var Description,
    all bidandbidder {
      high_bidder { var Buyername },
      bid {
        max ( all var Bid )
      }
    }
  }
}
FROM
and {
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        name { "Tom Jones" },
        userid { var Userid }
      }}
    }}
  },
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        offered_by { var Userid },
        var Description -> description {/.*Bicycle.*/},
        itemno { var Itemno }
```

```
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        itemno { var Itemno },
        userid { var Buyerid },
        bid { var Bid }
      }}
    }}
  },
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid { var Buyerid },
        name { var Buyername }
      }}
    }}
  }
}
END
```

3.1.4.5 Q9: Reguläre Ausdrücke für Selektion und Extraktion

List the number of items auctioned each month in 1999 for which data is available, ordered by month.

Die Datumsangaben sind in dem XML Dokument in der Form `YYYY-MM-DD` abgespeichert. Um aus dem Datum das Monat und das Jahr zu extrahieren und Wertevergleiche durchführen zu können, bietet XQuery die Funktionen `get-month-from-date()` und `get-year-from-date()` an.

```
<result>
  {
    let $end_dates := doc("items.xml")//item_tuple/end_date
    for $m in distinct-values(for $e in $end_dates
                              return get-month-from-date($e))
    let $item := doc("items.xml")
        //item_tuple[get-year-from-date(end_date) = 1999
                     and get-month-from-date(end_date) = $m]
    order by $m
    return
        <monthly_result>
            <month>{ $m }</month>
            <item_count>{ count($item) }</item_count>
        </monthly_result>
  }
</result>
```

Wie in der Anfrage von Abschnitt 3.1.4.1 erwähnt, sind in Xcerpt noch keine Datumstypen definiert und somit keine Funktionen auf Datumstypen implementiert. In Q1 dieses Anwendungsszenarios wurde das Vorhandensein der `current-date`-Funktion als gegeben vorausgesetzt. Dies wäre auch im Falle der beiden oben genannten Funktionen zu überlegen, allerdings

gibt es gute Gründe für einen anderen Lösungsweg. Wiedermal ist auf die Trennung zwischen Anfrage und Konstruktion zu achten. Während `current-date` eine Systemfunktion ist, die lediglich eine Art von atomaren Datenterm liefert, der immer anfragbar ist, erscheint die Extraktion von Monat und Jahr wie eine Konstruktion; nur wenn das Datum auch gebunden werden kann, kann man auch Teilinformationen daraus entnehmen. Würde man diese Anfrage mit diesen im Konstruktionsteil verwendeten Funktionen lösen, würde man auf jeden Fall zwei Regeln benötigen, da die Datumsangaben zuerst aufbereitet werden müssten.

Vielmehr bieten sich für die Extraktion reguläre Ausdrücke an, die dem Patternansatz entsprechen es weiterhin ermöglichen, mit nur einer Regel die Anfrage auszudrücken. Der reguläre Ausdruck setzt sich aus der expliziten Angabe des Jahres “1999”, eine Untergruppe für das Monat, welches an die Variable `Month` gebunden wird und einer Untergruppe für den Tag zusammen. Monat und Tag werden durch Charakterklassen genauer spezifiziert.

```
1  CONSTRUCT
2    result {
3      all monthly_result {
4        month { var Month },
5        item_count {
6          count ( all var Item )
7        }
8      } ordering [Month] literal
9    }
10 FROM
11   in {
12     resource [ "file:items.xml", "xml" ],
13     desc var Item -> item_tuple {{
14       end_date { /1999-(var Month::integer -> [01][0-9]-([0-3][0-9])/}
15     }}
16   }
```

3.1.4.6 Q10: Explizite Typumwandlung

For each item that has received a bid, list the item number, the highest bid, and the name of the highest bidder, ordered by item number.

Die Anforderungen dieser Anfrage sind recht einfach zu verwirklichen.

- **Regel 1 (Zeile 1-20):** Mit der Annahme, dass jedes Item, auf das geboten wurde, in der Datei “bids.xml” stehen muss, liefert diese Regel das höchste Gebot eines Items.
- **Regel 2 (Zeile 22-52):** Es wird ein Join über “users.xml” und “bids.xml” durchgeführt, um für jedes einzelne Gebot den Bieter zu erhalten.
- **Regel 3 (Zeile 54-78):** Die Ergebnisse der ersten beiden Regeln werden über `itemno` “gejoint”. Darüber hinaus wird ein Wertevergleich über `high_bid` und `bid` durchgeführt, um nur die Items mit dem höchsten Gebot zu erhalten.
- **Besonderheit:** Da die Höhe der Gebote in den XML Dokumenten ohne Typinformation abgespeichert sind (wie auch alle anderen Daten), müssen die Gebote einen einheitlichen Typ haben. In Zeile 6 wird eine Aggregation auf `Bid` durchgeführt, die das höchste Gebot ermitteln soll. Daher werden die Gebote zuerst in `float` umgewandelt. Damit

das höchste Gebot mit allen anderen in der dritten Regel verglichen werden kann, werden alle Gebote als `float` behandelt (Zeile 26, 58, 67 und 74). Der Programmierer hat hierbei auf die Korrektheit der Typbezeichnungen zu achten.

```
1  CONSTRUCT
2  high_bids {
3    all high_bid_tuple {
4      itemno { var Itemno },
5      high_bid {
6        max ( all toFloat( var Bid ))
7      }
8    }
9  }
10 FROM
11 in {
12   resource { "file:bids.xml", "xml" },
13   bids {{
14     bid_tuple {{
15       itemno { var Itemno },
16       bid { var Bid }
17     }}
18   }}
19 }
20 END
21
22 CONSTRUCT
23 bids {
24   all bid_tuple_with_username {
25     itemno { var Itemno },
26     bid { toFloat( var Bid ) },
27     name { var Name }
28   }
29 }
30 FROM
31 and {
32   in {
33     resource { "file:bids.xml", "xml" },
34     bids {{
35       bid_tuple {{
36         itemno { var Itemno },
37         bid { var Bid },
38         userid { var Userid }
39       }}
40     }}
41   },
42   in {
43     resource { "file:users.xml", "xml" },
44     users {{
45       user_tuple {{
46         userid { var Userid },
47         name { var Name }
48       }}
49     }}
50   }
51 }
52 END
53
54 CONSTRUCT
55 result {
56   all high_bid {
57     itemno { var Itemno },
```

```
    bid { var Bid::float },
    bidder { var Name }
  } ordering [Itemno] literal
}
FROM
and {
  bids {{
    bid_tuple_with_username {{
      itemno { var Itemno },
      bid { var Bid::float },
      name { var Name }
    }}
  }},
  high_bids {{
    high_bid_tuple {{
      itemno { var Itemno },
      high_bid { var Bid::float }
    }}
  }}
}
END
```

3.1.4.7 Q16: Bedingte Konstruktion

List all registered users in order by userid; for each user, include the userid, name, and an indication of whether the user is active (has at least one bid on record) or inactive (has no bid on record).

Die Besonderheit dieser Anfrage liegt in der Konstruktion eines `status`-Elements, dessen Inhalt davon abhängt, ob ein Teilnehmer einer Auktion geboten hat oder nicht. In XQuery ist dies einfach zu realisieren, da innerhalb der Konstruktion bedingte Anweisungen erlaubt sind.

```
<result>
  {
    for $u in doc("users.xml")//user_tuple
    let $b := doc("bids.xml")//bid_tuple[userid = $u/userid]
    order by $u/userid
    return
        <user>
            { $u/userid }
            { $u/name }
            {
                if (empty($b))
                  then <status>inactive</status>
                  else <status>active</status>
            }
        </user>
  }
</result>
```

In Xcerpt kann man, wie in Abschnitt 2.8.5 beschrieben, solche bedingten Konstruktionsanweisungen anhand von zwei sich ausschließenden Regeln verwirklichen. Man muss also sicherstellen, dass immer genau ein Regelrumpf mit einem Datenterm matcht und somit die zugehörige Konstruktion erfolgt. Diese Lösung ist sauber, hat aber häufig den Nachteil, dass

sich Anfrageteile und Konstruktionsteile nur minimal unterscheiden und somit viel redundante Schreibarbeit geleistet werden muss.

Allerdings gibt es, unter Verwendung von "bedingten Anfragen" zwei Möglichkeiten das Programm kompakter zu schreiben. Betrachte man zunächst folgendes Programm.

```
CONSTRUCT
   var State
FROM
   var State -> state {
       status {"active"},
       status {"inactive"}
   }
END

CONSTRUCT
result {
  all user {
    userid {var Userid},
    name {var Name}
  }
}
FROM
and {
  in {
    resource { "file:../users.xml", "xml" },
    users {{
      user_tuple {{
        userid {var Userid},
        name {var Name}
      }}
    }}
  },
  in {
    resource { "file:../bids.xml", "xml" },
    bids {{
      bid_tuple {{
        IF
          userid {var Userid}
        THEN
          state {{
            var Status -> status {"active"}
          }}
        ELSE
          state {{
            var Status -> status {"inactive"}
          }}
       }}
    }}
  }
}
```

Die erste Regel dient als eine Art "Zustandsdatenbank". Im Query Teil wird lediglich die Variable `State` an einen Term `state`, der unterschiedliche Zustandswerte enthält, gebunden. Durch die Ausgabe der Variable im Konstruktionsteil wird dieser Term den Datentermen hinzugefügt und kann deshalb durch weitere Regeln befragt werden. Im Anfrageteil der zweiten Regel wird innerhalb einer "bedingten Anfrage" überprüft, ob es für jeden Benutzer, der in der Datei "users.xml" gefunden wurde auch einen Eintrag in "bids.xml" gibt. Ist dies der Fall,

wird der neu erzeugte Datenterm nach `status{äctive"}` befragt und gebunden. Scheitert der Anfrageterm im `THEN`-Teil, wird `Status` an `status{ïnactive"}` gebunden. Die Benutzung einer solchen “Zustandsdatenbank” bietet sich besonders dann an, wenn es viele neue Terme gibt, die in die Ausgabe mit einfließen sollen. Ist die Datenmenge allerdings sehr groß, liegt es nahe, ein separates XML Dokument zu verwenden.

Eine andere und kürzere Lösungsmöglichkeit für diese Anfrage ist die Verwendung der Unifikation. In diesem Fall wird die Variable `Status` innerhalb des `THEN`- und `ELSE`-Teils mit dem gewünschten Term unifiziert, was einer expliziten Bindung entspricht.

```
CONSTRUCT
result {
  all user {
    userid {var Userid},
    name {var Name},
    var Status
  }
}
FROM
and {
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid {var Userid},
        name {var Name}
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        IF
          userid {var Userid}
        THEN
          var Status :< status {"active"}
        ELSE
          var Status :< status {"inactive"}
       }}
    }}
  }
}
```

3.1.5 Use Case “SGML: Standard Generalized Markup Language”

Das Dokument “report.xml” in Abschnitt 5.1.5.1 enthält einen Bericht, dessen Struktur durch Kapitel, Abschnitten, Titel und Paragraphen geprägt ist. Die Paragraphen enthalten den eigentlichen Text, der zusätzlich mit Markupinformationen über das Layout angereichert ist. Im folgenden werden Anfragen gestellt, die die Reihenfolge der Elemente anfragen, Zeichenfolgen im Text suchen und Wertevergleiche über Attribute durchführen.

3.1.5.1 Q4: Schrittweise Transformation

Locate the second paragraph in the third section in the second chapter (the second "para" element occurring in the third "section" element occurring in the second "chapter" element occurring in the "report").

Diese Anfrage kann man auf zwei Arten mit unterschiedlichen Funktionen lösen.

Nach dem Motto "Der Zweck heiligt die Mittel" entspricht die erste Lösung einer recht naiven Verarbeitungsweise der Daten und benutzt hierzu die Funktionen `first` und `drop N`. Hierbei werden die Daten schrittweise in Regeln selektiert. Zuerst wird das gesamte zweite Kapitel selektiert. Anschließend wird in diesem Kapitel der dritte Abschnitt ausgegeben und zuletzt in diesem Abschnitt der gesuchte zweite Paragraph selektiert und ausgegeben. Diese Art der Anfrage ist aber nur dann praktikabel, wenn man eine relativ flache Struktur anfragt, da für jede Ebene eine extra Regel geschrieben werden muss.

```
1  CONSTRUCT
2    temp1 [
3      first (drop 1 (all var Chapter ))
4    ]
5  FROM
6    in {
7      resource { "file:report.xml", "xml" },
8      report [[
9        var Chapter -> chapter [[]]
10     ]]
11   }
12 END
13
14 CONSTRUCT
15   temp2 [
16     first ( drop 2 (all var Section ))
17   ]
18 FROM
19   temp1 [[
20     desc var Section -> section {{}}
21   ]]
22 END
23
24 CONSTRUCT
25   result [
26     first (drop 1 ( all var Para ))
27   ]
28 FROM
29   temp2 [[
30     desc var Para -> para {{}}
31   ]]
32 END
```

Die zweite und sicherlich allgemeinere Lösung benötigt eine Regel zur Vorbereitung der Daten, um den gewünschten Paragraphen in einer zweiten Regel über ein Pattern mit Positionsangaben zu selektieren. Die Vorbereitungsregel selektiert alle `chapter`-, `section`- und `para`-Terme und gibt diese unter Beibehaltung der Beziehungen in einer geschachtelten Termstruktur wieder aus. Die Erhaltung der Beziehung erreicht man, indem die jeweiligen Variablenbindungen innerhalb neuer Terme (`meta_chapter`, `meta_section` und `meta_para`) ausgegeben werden. Man kann diese neuen Terme als eine Art "Metaterme" sehen. Jeder Metaterm enthält seinen

entsprechenden Term und die dazugehörigen Subterme, die wiederum innerhalb ihrer Metaterme stehen. Indem man das `all`-Konstrukt vor den Metatermen verwendet und nicht vor den Variablen, bleiben die Vater-Kind Beziehungen erhalten.

```
CONSTRUCT
temp [
  all meta_chapter [
    var Chapter,
    all meta_section [
      var Section,
      all meta_para [
        var Para
      ]
    ]
  ]
]
FROM
in {
  resource { "file:report.xml", "xml" },
  report [[
    var Chapter -> chapter [[
      var Section -> section [[
        var Para -> para {{}}
      ]]
    ]]
  ]]
}
END

CONSTRUCT
  result [
    var Para
  ]
FROM
  temp [[
  position 2 meta_chap [[
    position 4 meta_section [[
      position 3 meta_para {{ var Para }}
  }}
  ]]
END

CONSTRUCT
  result [
    first (drop 1 ( all var Para ))
  ]
FROM
  temp2 [[
    desc var Para -> para {{}}
  ]]
END
```

3.1.5.2 Q7: Positionelle Anfrage mit Aggregationsfunktionen

Locate the initial letter of the initial paragraph of all introductions (the first character in the content [character content as well as element content] of the first ”para” element contained in an ”intro” element).

Zuerst werden die ersten Paragraphen aller Einleitungen ermittelt. Hierzu werden die `intro`-Bezeichner durch einen regulären Ausdruck an eine Variable gebunden werden, um die Beziehung zwischen den "Paragraphen" und den "Introductions" im Konstruktionsteil beizubehalten. Durch die geschachtelten `all`-Konstrukte erhält man für jeden `intro`-Bezeichner den ersten `para`-Term. Die Beziehung könnte auch über "Metaterme", wie sie in der Anfrage Q4 verwendet werden, erhalten werden. Die zweite Regel bindet anhand eines regulären Ausdrucks die ersten Zeichen aller Unterelemente von `para` an die Variable `FirstLetter`. Allerdings ist man nur an der ersten Variablenbindung eines jeden Paragraphens interessiert. Daher bindet man den Bezeichner `para` ebenfalls an eine Variable und gibt im Konstruktionsteil das erste gefundene Zeichen jedes Paragraphens aus.

```
1  CONSTRUCT
2  temp [
3    all var Intro [
4      first ( all var Para )
5    ]
6  ]
7  FROM
8  in {
9    resource { "file:report.xml", "xml" },
10   desc /(var Intro -> intro)/ [[
11     var Para -> para [[]]
12   ]]
13 }
14 END
15
16 CONSTRUCT
17 result [
18   all var Para {
19     first_letter {
20       first (all var FirstLetter )
21     }
22   }
23 ]
24 FROM
25 temp [[
26   desc /(var Para -> para)/ [[
27     desc /(var FirstLetter -> .).*/
28   ]]
29 ]]
30 END
```

3.1.5.3 Q8a: Beseitigung von Markup

Locate all sections with a title that has "is SGML" in it. The string may occur anywhere in the descendants of the title element, and markup boundaries are ignored.

In diesem Beispiel muss das Markup mit der Funktion `toText()` entfernt werden. In der nötigen Regel zur Vorbereitung der Daten müssen sowohl die `section`-Terme wie auch der Inhalt von `title` gebunden werden, und als Sequenz innerhalb eines neuen Terms ausgegeben werden um die Beziehung der beiden Terme zu erhalten. In der zweiten Regel werden anschließend alle Abschnitte gebunden, die den betreffenden Substring enthalten.

```
CONSTRUCT
  temp [
    all section_title {
      var Section,
      content {
        toText (var Content)
      }
    }
  ]
FROM
  in {
    resource { "file:report.xml", "xml" },
    desc var Section -> section [[
      title {{ var Content }
    ]]
  }
END

CONSTRUCT
  result [ all var Section ]
FROM
  section_title [[
    var Section -> section {{}},
    content {{ /.*is SGML.*/
  ]]
END
```

3.1.5.4 Q9: Joins über Attribute

Locate all the topics referenced by a cross-reference anywhere in the report (all the ”topic” elements whose ”topicid” attribute value is the same as an ”xrefid” attribute value of any ”xref” element).

Attribute werden in Xcerpt bis auf Ordnung in der gleichen Weise betrachtet wie Elemente. So wird ein Join über zwei Attribute durchgeführt, indem die entsprechende Anfrageterme innerhalb eines `and`-Konstruktes spezifiziert werden, und die Werte der Attribute `xrefid` und `topicid` über gleiche Variablennamen verglichen werden.

```
CONSTRUCT
  result [ all var Topic ]
FROM
  in {
    resource { "file:report.xml", "xml" },
    and {
      desc var Topic -> topic [[
        attributes  {{
          topicid [ var XRefID ]
        }}
      ]],
      desc xref [[
        attributes  {{
          xrefid [ var XRefID ]
        }}
      ]]
    }
  }
END
```

3.1.6 Use Case "STRING: String Search"

Dieser Anwendungsfall basiert auf zwei Dokumenten (Abschnitt 5.1.6.1). Das erste ("company_data.xml") beinhaltet Firmenprofile, das zweite ("news.xml") ist ein Nachrichtendokument, welches Informationen über Öffentlichkeitsarbeit, Fusionen und Akquisitionen enthält. Viele der Anfragen benötigen Textsuche über Elementnamen und deren Inhalt. Weiterhin wird gezeigt, wie man in einem Dokument Zeichenketten selektiert und versucht, diese im Text des zweiten Dokuments zu finden.

3.1.6.1 Q2: Wertevergleich in regulären Ausdrücken

Find news items where the Foobar Corporation and one or more of its partners are mentioned in the same paragraph and/or title. List each news item by its title and date.

Das Besondere an dieser Anfrage ist das selektieren der Partner in einem Dokument und das gleichzeitige Auffinden der Partner innerhalb eines Textes eines anderen Dokumentes. Hierzu wird ein Join verwendet, der durch die `and`-Gruppierung um die beiden Anfrageterme (jeweils eingeleitet durch `in{}`) der ersten Regel und durch die Verwendung der gemeinsame Variable `Partner` stattfindet. Es ist offensichtlich, dass in dem ersten Anfrageterm alle Partner der "Foobar Corporation" an die Variable `Partner` gebunden werden. Interessanter hingegen ist die Variablendeklaration im zweiten Term innerhalb der regulären Ausdrücke (Zeile 22 und 26). Für das Patternmatching in regulären Ausdrücken werden so genannte Charakterklassen benutzt, dargestellt durch die eckigen Klammern. In Charakterklassen kann man definieren, welche Zeichen an der spezifizierten Stelle vorkommen dürfen. In diesem Beispiel wird also die Variable `Partner` an alle Substrings der Inhalte von `title` und `par` gebunden, die mit einem oder zwei Wörtern matchen. Wörter werden über Charakterklassen spezifiziert; ein Wort beginnt mit einem Großbuchstaben (alle Zeichen von "A bis Z"), gefolgt von einer beliebigen Menge von Klein- und Großbuchstaben. "Beliebig viele" wird mit dem "Kleenestar" nach der Charakterklasse ausgedrückt, die Optionalität des zweiten Wortes mit "?". All diese möglichen Variablenbindungen aus den regulären Ausdrücken werden mit den Bindungen, die im ersten Anfrageterm gefunden wurden, verglichen. Gibt es wenigstens eine Übereinstimmung, so war der Join erfolgreich und die Query ebenfalls (mit der Voraussetzung, dass auch die "Foobar Corporation" in `title` oder `par` enthalten war). Es wird deutlich, dass diese Art der Verwendung von regulären Ausdrücken noch nicht optimal ist. In diesem Beispiel geht man davon aus, dass ein Firmenname aus höchstens zwei Wörtern besteht, was aber in der Praxis nicht allgemein gültig ist. Würde man aber einen regulären Ausdruck angeben, der über über den gesamten Inhalt matchen kann würde man kein Ergebnis erhalten. Ein regulärer Ausdruck nach dem POSIX Standard liefert, abhängig von der Art der Implementierung der regulären Ausdrücke, nämlich nur den ersten Treffer. In Perl wird beispielsweise nur der größtmögliche String geliefert, der auf den regulären Ausdruck passt. In Xcerpt wird eine Implementierung der regulären Ausdrücke verwendet werden, die alle möglichen "Matches" liefern.

In dieser Regel ist es allerdings nicht möglich, den Term `title` an eine Variable zu binden, da nicht gewährleistet werden kann, dass die Und-Verknüpfung (Zeile 20-23) erfolgreich ist. Deswegen wird der gesamte `news-item`-Term gebunden und in der zweiten Regel die gewünschten Elemente selektiert und ausgegeben.

```
1  CONSTRUCT
2  temp {
3    all news_item {
4      var News
5    }
6  }
7  FROM
8  and {
9    in {
10     resource { "file:company_data.xml", "xml" },
11     company {{
12       name {"Foobar Corporation"},
13       desc partner {var Partner}
14     }}
15   },
16   in {
17     resource { "file:input.xml", "xml" },
18     desc news_item {{
19       or {
20         and {
21           title {/.*Foobar Corporation.*/},
22           title {/.*(var Partner ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/}
23         },
24         and {
25           desc par {/.*Foobar Corporation.*/},
26           desc par {/.*(var Partner ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/}
27         }
28       }}
29     }
30   }
31 }
32 END
33
34 CONSTRUCT
35 result {
36   all news_item {
37     var Title,
38     var Date
39   }
40 }
41 FROM
42 temp {{
43   news_item {{
44     var Title -> title{{}},
45     var Date -> date {{}}
46   }}
47 }}
48 END
```

3.1.6.2 Q4: Wertevergleich über Variablen in regulären Ausdrücken

Find news items where a company and one of its partners is mentioned in the same news item and the news item is not authored by the company itself.

Diese Anfrage basiert auf dem selben Ansatz wie Q2 dieses Anwendungsszenarios. Allerdings muss noch zusätzlich beachtet werden, dass die Firma, die in “company_data.xml” und in dem Inhalt von `news_item` gefunden wird, nicht innerhalb des `news_agent`-Term auftritt. Diese Bedingung erreicht man durch eine “Term Negation” in Zeile 22. Somit wird die Variable

`Company` nur an die Literale des Terms `news_agent` gebunden, die sich von den möglichen Literalen des Terms `news_item` unterscheiden.

```
CONSTRUCT
result {
  all var News_Item
}
FROM
and {
  in {
    resource { "file:company_data.xml", "xml" },
    company {{
      name {var Company},
      desc partner {var Partner}
    }}
  },
  in {
    resource { "file:input.xml", "xml" },
    and {
      desc var News_Item -> news_item {{
        and {
          desc /.*(var Company ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/,
          desc /.*(var Partner ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/
        },
        without news_agent{ var Company }
      }}
    }
  }
}
END
```

3.1.7 Use Case “NS - Queries Using Namespaces”

Wie in Abschnitt 2.8.7 angemerkt ist der Namensraummechanismus in Xcerpt noch nicht vollständig ausgereift (vgl. Verschachtelung und Überschattung). Diese Eigenschaften sind aber vor allem für die Transformation von Dokumenten wichtig. Die Anfragen dieses Anwendungsfalls sind allerdings geprägt durch Selektionen von Namensräumen und durch Wertevergleiche, was mit der Sprache Xcerpt in ihrem momentanen Entwicklungszustand gut zu verwirklichen ist. Das verwendete Dokument in Abschnitt 5.1.7.1 beinhaltet Auktionen verschiedener Online-Auktionshäuser mit Informationen über das Produkt, Beginn und Ende einer Auktion, dem Benutzer mit dem höchsten Gebot und dem Anbieter.

3.1.7.1 Q3: Reguläre Ausdrücke in Namensräumen

Select all elements that have an attribute whose name is in the XML Schema namespace.

Das Suchpattern beschreibt einen Term, dessen Bezeichner sich aus einem beliebigen Namensraum und einem beliebigen Bezeichner zusammensetzt, aber mindestens ein Attribut besitzt, welches den Namensraum ”http://www.w3.org/2001/XMLSchema”hat.

```
ns-prefix dt="http://www.w3.org/2001/XMLSchema"

CONSTRUCT
Q3 {
  all var Element
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc var Element -> /.*/:/.*/ {{
    attributes {{
      dt:/.*/ {{}}
    }}
  }}
}
END
```

3.1.7.2 Q7: Variablen für Namensräume

Select the homepage of all auctions where both seller and high bidder are registered at the same auctioneer.

In Xcerpt können Namensräume an Variablen gebunden werden. Um einen gemeinsamen Namensraum zu selektieren, der aber durch unterschiedliche Namensraumpräfixe vertreten wird, wird anstellen der Präfixe eine gemeinsame Variable geschrieben und darüber ein Join ausgeführt.

```
ns-prefix ma="http://www.example.com/AuctionWatch"

CONSTRUCT
Q7 {
  all var Homepage
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc  ma:Auctions {{
    var Homepage -> ma:AuctionHomepage {{}},
    and {
      desc ma:HighBidder {{
        var NS:ID {{}}
      }},
      desc ma:Seller {{
        var NS:ID {{}}
      }}
    }
  }}
}
END
```

3.1.8 Use Case “PARTS - Recursive Parts Explosion”

In dieser Anwendungsfall wird gezeigt, wie aus einer flachen Liste von Elementen, die durch Referenzen hierarchisch miteinander verknüpft sind die entsprechende Baumstruktur aufgebaut wird. Es werden zwei Lösungen vorgestellt; die eine baut die Struktur über rekursive

Regeln auf. Die zweite bedient sich des Xcerpt-Referenzmechanismus und benötigt hierzu nur eine einfache Transformationsregel. Das Dokument ist im Anhang in Abschnitt 5.1.8.1.

3.1.8.1 Q1: Umwandlung einer Liste in eine Baumstruktur

Convert the sample document from "partlist" format to "parttree" format (see DTD section for definitions). In the result document, part containment is represented by containment of one `<part>` *element inside another. Each part that is not part of any other part should appear as a separate top-level element in the output document.*

Dies ist ein weiteres Beispiel für die Verwendung von rekursiven Regeln. Im Gegensatz zu der Anfrage im Anwendungsfall "TREE" wird nicht aus einem Eingabebaum ein neuer geschaffen. Die Elemente (welche alle den Bezeichner `part` besitzen) sind als Liste innerhalb eines Wurzelelements abgespeichert und über ID's verknüpft. Über diese ID-Attribute (`partid` und `partof`) wird eine hierarchische Struktur beschrieben. Diese Baumstrukturen sollen durch die Anfrage aufgebaut werden. Dazu reichen zwei Regeln aus. Die erste Regel (Zeile 13-44) matcht mit allen Wurzelelementen und gibt diese wieder aus. Wenn es in der Liste allerdings noch Elemente gibt, die Kinder dieses Wurzelelementes sind, wird versucht, das Prädikat `hasChild` mit dem Kind als Eingabeargument zu erfüllen. Das "wenn" wird durch ein `optional` vor den Anfragetermen in Zeile 35 und 42 ausgedrückt. Beim dem Versuch, das Prädikat `hasChild` zu erfüllen, wird die zweite Regel angewandt. Die Attribute `partid` und `partof` des Eingabeterms werden gebunden und es wird überprüft, ob es in der Liste ein Element gibt, das ein Unterteil der Eingabe ist. Gibt es ein oder mehrere Unterteile, wird `hasChild` rekursiv für jedes dieser Unterteile angewandt und die Ergebnisterme in die Ausgabe mit aufgenommen (Zeile 54). Gibt es keine Unterteile, wird lediglich der Eingabeterm zurückgegeben.

```
1  GOAL
2  var Result
3  FROM
4  and {
5    in {
6      resource { "file:partlist.xml", "xml" },
7      var Root -> partlist {{}}
8    },
9    init [ var Root, var Result ]
10 }
11 END
12
13 CONSTRUCT
14 init [
15   var In,
16   result [
17     all part {
18       attributes {
19         partid { var PartId },
20         name { var Name }
21       },
22       optional all var ChildPartRes
23     }
24   ]
25 ]
26 FROM
27 and {
28   partlist {{
```

```
    part {
      attributes {
        partid { var PartId },
        name { var Name }
      }
    },
    optional var ChildPart -> part {
      attributes {
        partid {{}},
        partof { var PartId }
      }
    }
  }} :< var In,
  optional hasChild [ var ChildPart, var ChildPartRes ]
}
END

CONSTRUCT
hasChild [
  var In,
  part {
    attributes {
        partid { var Partid },
        name { var Name }
    },
    optional all var ChildPartRet
  }
]
FROM
and {
  part {{
    attributes {
      partid { var Partid },
      partof { var Partof },
      name { var Name }
    }
  }} :< var In,
    in {
    resource { "file:partlist.xml", "xml" },
    optional partlist {{
      var Child -> part {{
        attributes {
          partid {{}},
          partof { var Partid  }
        }
      }}
    }}
  },
  optional hasChild [ var Child, var ChildPartRet ]
}
END
```

Eine zweite Lösung macht sich den Referenzmechanismus von Xcerpt zu nutze um die gewünschte Baumhierarchie mit nur einer Regel zu erzeugen. Diese fügt allen `part`-Elementen eine Xcerpt-ID und allen Elementen, die Kinderelemente besitzen, Xcerpt-Referenzen auf alle diese Kinder hinzu.

Hierzu werden alle `part`-Elemente im Anfrageteil selektiert und der Inhalt des `partid`-Terms an eine Variable gebunden. Im Konstruktionsteil werden alle Teile mit ihrem entsprechenden ID-Literal, gefolgt von dem ID-Symbol `@`, versehen. Die Referenzen auf die Kindelemente

liefert der als `optional` gekennzeichnete Anfrageterm (Zeile 21-27), welcher jedes Teil liefert, deren `partof`-Element die ID des ersten Anfrageterms beinhalten. Die ID dieses Teils wird an die Variable `PartId2` gebunden und im Konstruktionsterm innerhalb des Vaterelements zusammen mit einem Referenzsymbol `^` angefügt.

Das Ergebnis in Xcerpt-Syntax wäre dann eine Liste von Teilen, die durch den ID/IDREF-Mechanismus von Xcerpt implizit verknüpft sind. Gibt man das Ergebnis allerdings in XML Format aus, erhält man die gewünschte Baumstruktur.

```
1  CONSTRUCT
2  parttree {
3    all var PartId1@part {
4      attributes {
5        name { var Name }
6      },
7      optional all ^var PartId2
8    }
9  }
10 FROM
11 in {
12   resource { "file:partlist.xml", "xml" },
13   partlist {{
14     part {
15       attributes {
16         partid { var PartId1 },
17         optional partof { var Partof },
18         name { var Name }
19       }
20     },
21     optional part {
22       attributes {
23         partid { var PartId2 },
24         partof { var PartId1 },
25         name { var Name2 }
26       }
27     }
28   }}
29 }
30 END
```

3.1.9 Use Case “STRONG - Queries That Exploit Strongly Typed Data”

In diesem Anwendungsszenario werden Daten verwendet die einem XML Schema [DCF01] unterliegen um die “Simple Types” und “Strong Types” von XML Schema zu verwenden. Die Anfragen sind darauf ausgelegt, Elemente der Datenbank auf Konformität mit dem Schema zu überprüfen. Ein fundiertes Typsystem von Xcerpt befindet sich noch in der Entwicklungsphase. Zum Momentanen Zeitpunkt ist es allerdings noch nicht möglich XML Schema und DTD’s zu befragen.

3.2 Use Case: Der Referenzmechanismus von Xcerpt

Referenzen sind ein notwendiges Mittel zur Vermeidung und Auflösung von Redundanzen. Der in XML hierfür vorgesehene ID/IDREF-Mechanismus verwendet in den Dokumentinstanzen Attribute, welche mit den Typen `id` und `idref` versehen sind. Diese Typen werden in dem zugehörigen XML-Schema für die entsprechenden Attribute deklariert (siehe Abschnitt 2.2).

In Xcerpt hingegen wird ein Referenzmechanismus verwendet, der ohne Schema arbeitet; die Referenzen werden implizit von Xcerpt aufgelöst. Die aus den Referenzen resultierende Graphstruktur kann somit direkt angefragt werden.

Betrachte man folgenden Ausschnitt eines XML Dokuments, welches ein Buch mit einem Titel beinhaltet, gefolgt von Autorenelementen, deren Attribut `idref` auf Autoren innerhalb von `authorlist` zeigen.

```
<root>
  <book>
    <title>Deductive and Object-Oriented Databases</title>
    <author idref="a1"/>
    <author idref="a2"/>
    <author idref="a3"/>
  </book>
  <authorlist>
    <name id = "a1">François Bry</name>
    <name id = "a2">Raghu Ramakrishnan</name>
    <name id = "a3">Kotagiri Ramamohanarao</name>
  <authorlist>
</root>
```

Eine Anfrage, die alle Bücher mit Titel und all ihren Autoren ausgeben soll, muss einen Join über die `id`/`idref`-Attribute durchführen, da selbst XML-Anfragesprachen, die den XML-Referenzmechanismus unterstützen würden, ohne ein Schema keine implizite Dereferenzierung durchführen können. Eine Xcerpt-Anfrage auf das obige XML-Dokument, die Bücher mit ihren Autoren ausgibt lautet:

```
CONSTRUCT
  result {
    all book {
      var Title,
      all var Name
    }
  }
FROM
  and {
    book {{
      var Title -> title {{}},
      author {
        attributes {
          idref { var ID }
        }
      }
    }},
    authorlist {{
      var Name -> name {
        attributes {
          id { var ID }
        }
```

```
      }
    }}
  }
END
```

Wird das XML-Dokument allerdings in Xcerpt-Datentermen und mithilfe des Xcerpt-Referenzmechanismus dargestellt, reicht in der Anfrage eine simple Selektion der Bücher aus. Man muss keine neue Struktur aufbauen, da diese implizit schon vorhanden ist.

Darstellung als Xcerpt-Datenterm:

```
root {
  book {
    title { "Deductive and Object-Oriented Databases" },
    ^a1,
    ^a2,
    ^a3
  },
  a1@author { "François Bry" },
  a2@author { "Raghu Ramakrishnan" },
  a3@author {"Kotagiri Ramamohanarao" }
}
```

Xcerpt-Anfrage, die nur die Bücher selektiert:

```
CONSTRUCT
  result {
    all var Book
  }
FROM
  root {{
    var Book -> book {{}}
  }}
END
```

Es ist zu beachtet, dass sich ein syntaktischer Unterschied daraus ergibt, da Xcerpt-Referenzen nicht nur als Attribute definiert werden können, sondern für jeden beliebigen Term stehen können. Betrachte man das oben vorgestellte XML-Dokument. Jedes `idref`-Attribut tritt innerhalb eines `author`-Elements auf und jedes `id`-Attribut eines Autors innerhalb eines `name`-Elements. Dies ist aus zwei Gründen nötig. Zum einem dürfen Attribute gleichen Namens nicht innerhalb eines Elements stehen. Wenn aber jedes Attribut für einen Autor einen anderen Namen besitzen würde und man die Referenzen auflösen möchte, hätte man keinen Einfluss auf die Reihenfolge der dereferenzierten Elemente. Die Menge der Attribute eines Elements ist als eine unsortierte definiert. Will man die Reihenfolge bestimmen, muss jedes `idref`-Attribut, wie in dem XML-Dokument gezeigt, innerhalb eines eigenen `author`-Elements erscheinen. Löst ein XML-Prozessor die Referenzen nun auf, würde man als Ergebnis immer das Zwischenelement `name` erhalten, welches den eigentlichen Namen enthält. Dies ist kein Nachteil, muss aber bei einem Entwurf eines Datenmodells berücksichtigt werden. In Xcerpt kann dieses "Zwischenelement" vermieden werden, da die Referenz direkt an der Stelle eines Xcerpt-Terms angegeben werden kann, wie in dem obigen Xcerpt-Datenterm gezeigt.

3.3 Use Case: Webcrawler

Dieses Anwendungsszenario benutzt Xcerpt-Regeln, um das Web rekursiv zu durchlaufen. Das Programm sammelt hierzu, ausgehend von einer Startwebseite, alle Hyperlinks dieser Seite auf und setzt eine Suche auf allen Seiten, auf welche die Links zeigen, fort. Programme die diese Funktionalität besitzen, nennt man auch “Webcrawler”.

Um einen Webcrawler in Xcerpt zu implementieren, benötigt die Anfragesprache eine Erweiterung, die im momentanen Xcerpt-Prototypen noch nicht eingebaut ist, in einer späteren Version aber zur Verfügung stehen wird. So muss es möglich sein, Variablen in den `resource`-Termen zu deklarieren, da die gefundenen Hyperlinks einer Web-Seite in einem rekursiven Aufruf einer Xcerpt-Regel zur Angabe der Ressourcenspezifikation verwendet werden müssen. Im folgenden wird die Deklarationsmöglichkeit der bisher vorgestellten “Variablen ohne Einschränkung” in `resource`-Termen angenommen.

Wie in den Anwendungsfällen in den Abschnitten 3.1.2.1 und 3.1.8.1, wo hierarchische Strukturen in unbestimmter Tiefe durchsucht werden, wird in diesem Programm, welches eine potentiell unbegrenzte Graphstruktur durchläuft, rekursive Regeln benutzt.

Da das Web einen Graphen repräsentiert und daher Zyklen vorkommen, ist die Terminierung eines solchen Webcrawlerprogramms von besonderer Bedeutung. Ohne Terminierung wird das Web in ein unendliches Dokument übersetzt. In diesem Anwendungsbeispiel wird davon ausgegangen, dass das Programm aber nur dann terminiert, wenn keine weiteren Hyperlinks gefunden werden können. Zirkuläre Pfade werden nicht erkannt und deshalb immer wieder durchlaufen.

Xcerpt-Programm: Webcrawler

```
CONCTRUCT
  var The_Web
FROM
  recursion [
    url_in {"www.pms.informatik.uni-muenchen.de"},
    linktree {
      var The_Web
    }
  ]
END

CONSTRUCT
  recursion [
    url_in { var Resource },
    linktree {
      all link {
        var Resource ,
        optional all link { var Link2 }
      }
    },
  ]
FROM
  and {
    in {
      resource { var Resource, "html" },
      optional desc {{
        attributes {{ href { var Link1 } }}
      }}
```

```
29      },
30      optional recursion [
31        url_in { var Link1 },
32        linklist {{ var Link2 }},
33      ]
34    }
35  END
```

Zur Initialisierung dient eine Regel (Zeile 1-10), mit einem Prädikat `recursion` und seinen Kindtermen `url-in{}` und `linktree{}` im Anfrageteil. Der Inhalt von `url-in{}` ist der URL-String der Startseite, `linktree{}` enthält nach einer Terminierung oder nach einem möglichen Benutzerabbruch einen Ausschnitt der Linkstrukturen des WWW in Form eines Baumes. Die Darstellung der Linkstruktur als Graph könnte man mithilfe des Referenzmechanismus von Xcerpt darstellen, was aber im folgenden nicht weiter erläutert wird. Dies würde die Darstellung von zyklischen Referenzen ermöglichen.

Es wird also versucht, das Prädikat `recursion` zu erfüllen indem es auf den Kopf der zweiten Regel angewandt wird. Dadurch wird der Anfrageteil dieser Regel aufgerufen (Zeile 22-35). In der angegebene Quelldatei werden alle Attribute namens `href` gesucht und deren Werte (also URL's) an die Variable `Link1` gebunden. Für alle diese gefundenen URL's wird das Prädikat `recursion` mit der URL als Eingabewert aufgerufen. Die Terminierung erfolgt über die `optional`-Angaben. Werden nämlich keine `href`-Attribute auf einer Web-Seite gefunden, wird nur die Quell-URL in `linktree` geschrieben (Zeile 18, 26 und 30).

3.4 Use Case: Ein Mediator

3.4.1 Einführung

Durch die ständige Erschließung neuer Möglichkeiten im Bereich der Datenverarbeitung entstehen immer größere und zuweilen auch unüberschaubare Datenmengen, auf die zwar immer schneller zugegriffen werden kann, die immer schneller verarbeitet und transportiert werden können, aber auch immer komplexer und syntaktisch wie semantisch verschiedener werden.

Um mit solchen heterogen Datenquellen arbeiten zu können, sind Methoden der Datenintegration nötig, die aber nicht nur die vorhandenen Daten in einer bestimmten Weise zusammenfügen, sondern diese verarbeiten und in Wissen überführen und aufgrund dieses Wissens Entscheidungen getroffen werden können.

Der Bedarf an Systemen, die solche Vorgänge automatisieren können ist entsprechend hoch, wenn man die Bedeutung von Datenverarbeitung in größeren Unternehmen und Institutionen betrachtet.

Im folgenden wird deshalb der Ansatz von Mediatoren betrachtet, die zumindest ihrer Architektur nach solche Dienste zu leisten vermögen und die eine Anfragesprache benötigen, die mit heterogenen Daten umgehen kann. Wie der Name vermuten lässt, stellt ein solches Systeme eine Art Vermittler zwischen Datenquellen und den Anwendungen dar, der die Daten aufbereitet und in der gewünschten Form liefert.

Anhand der Anfragesprache Xcerpt wird ein vereinfachtes Anfragesystem vorgestellt, welches als ein simpler Mediator gesehen werden kann. Zunächst wird veranschaulicht, welche Probleme bei der Integration von heterogene Daten auftreten können.

3.4.1.1 Heterogene Datenquellen

Eine Vermittlung zwischen unterschiedlichen Datenquellen geschieht nicht allein über strukturierte Daten, wie es im besten Fall Datenbankmanagementsysteme liefern, sondern enorme Mengen an Texten, die so gut wie keine Metadaten oder auch besondere Strukturen besitzen (semi- oder unstrukturiert). Bei der Vereinigung dieser Vielzahl von unterschiedlichen Datenformaten kommt es zu mehreren Problemen.

- **Abstraktion:** Die Datenmenge muss durch eine geeignete Auswahl reduziert werden. Allerdings ist eine solche Auswahl nicht einfach, wenn es darum geht, aus den vorhandenen Daten Wissen zu generieren, das z.B. als Entscheidungshilfe dienen soll, wozu man unterschiedliche Methoden benötigt. Eine solche Methode könnte eine Berechnung sein, die aus Flugplänen die Reisedauer und die Kosten berechnet. Die Komplexität einer solchen Methode kann sehr hoch sein und übersteigt zumeist die Möglichkeit eines DBMS. Werden diese Methoden in Anwendungsprogrammen implementiert, so wird das Wissen verborgen und kann von anderen Anwendungen nicht genutzt werden. Daraus ergibt sich die Forderung nach einem Mediator.
- **Mangelnde Übereinstimmung:** Wie bereits erwähnt, gibt es bei Daten unterschiedlicher Herkunft Konflikte bezüglich Struktur, Repräsentation und Semantik. Benennungsunterschiede sind hierbei noch das kleinste Problem. So müssen bei der Dokumentvereinigung Aspekte wie der ortsgebundene Geltungsbereich von Dokumenten, die Gültigkeit bezüglich der Zeit und die interne Semantik der Dokumente bezüglich ihrer Werte berücksichtigt werden können. Die Auflösung solcher Probleme benötigt zumeist ein hohes Maß an Expertenwissen und eine gute Kenntnis über die Struktur der Daten.
- **Strukturiertheit von Datenquellen:** Datenquellen müssen nicht unbedingt strukturiert sein. Man unterscheidet hier strukturierte, semistrukturierte und unstrukturierte Daten. Es ist klar, dass auf unstrukturierte Daten keine typischen Datenbankanweisungen ausgeführt werden können. Dennoch ist es wichtig, dass auch solche Datenquellen in ein System integriert werden und in deskriptiver Weise angefordert werden können. Hierzu müssen Methoden des Information Retrieval angewendet werden, die diese Dokumente mit Struktur anreichern, um auch solche Daten verarbeiten und in vereinheitlichter Weise ausgeben zu können.

3.4.1.2 Anforderungen an Mediator

Aus den soeben vorgestellten Problem ergeben sich folgende Anforderungen an ein Mediator-System.

- **Deskriptive Anfragesprache:** Ein Benutzer muss beschreiben können, welche Informationen er aus den heterogenen Datenquellen benötigt und nicht, wie sie zu beschaffen sind. Aus diesem Grund wird eine deskriptive Anfragesprache benötigt. Hierbei ist zu beachten, dass sowohl genaue wie auch ungenaue Anfragen möglich sein müssen, da die darunter liegenden Datenquellen möglicherweise keine genauen Anfragen zulassen.
- **Kombination und Korrelation der Datenquellen:** Es muss möglich sein, die Daten der verschiedenartig strukturierten Quellen zu kombinieren und zu korrelieren. Hierbei sollen semantische Unterschiede und Redundanzen soweit wie möglich aufgelöst werden.

- **Abstraktion:** Die Menge der Daten muss durch geeignete Abstraktion reduziert werden können.
- **Einheitliche Darstellung der Ergebnisse:** Die Information, die das System liefert, muss in einer einheitlichen Weise dargestellt werden, unabhängig davon, wie die Ursprungsstruktur ist.

3.4.2 Xcerpt-Regeln für einen Mediator

Im folgenden wird gezeigt, wie heterogene Datenquellen mit Xcerpt-Regeln in eine vereinheitlichte Struktur überführt werden, und die Daten dieser Struktur mit einer Anfrage befragt wird. Die Funktionalität des Mediators besteht also darin, unterschiedliche Datenquellen zusammen zu führen und unterschiedlichen Anwendungen für Anfragen zur Verfügung zu stellen.

Die Datenquellen: In diesem Szenario werden drei Quelldateien benutzt, die wissenschaftliche Bücher und Artikel enthalten. Die Datei "dblp.xml" (zu finden in Abschnitt 5.2.1) beinhaltet bibliographische Informationen über Journale und Tagungen im Bereich der Informatik, die von dem DBLP-Server[1] zur Verfügung gestellt werden. Die Daten setzten sich aus Titeln von Veröffentlichungen, deren Autoren, dem Erscheinungsjahr und einer Angabe über die Seitenzahl zusammen. Die Datei "amazon.xml" (zu finden in Abschnitt 5.2.2) beinhaltet zu Anschauungszwecken alle Bücher mit dem Autor "François Bry", die in dem Onlinebuchladen "Amazon"[2] erhältlich sind. Die Datensätze beinhalten eine URL auf eine Informationsseite zu diesem Buch, einen Titel, Autoren, Erscheinungsdatum, die Lieferfirma und den Verkaufspreis. Die dritte Quelle "pms.xml" (Abschnitt 5.2.3) ist ein Ausschnitt einer Web-Seite des Informatik-Lehrstuhls für Programmier- und Modellierungssprache der LMU[3], welche die Veröffentlichungen dieses Lehrstuhls beinhaltet. Die Datei ist im HTML-Format und beinhaltet neben den Titeln der Veröffentlichungen deren Autoren und Verweise auf diese Dokumente, die zumeist im PDF-Format abgespeichert sind. Alle Quelldateien sind für ein einfacheres Verständnis in ihrem Inhalt stark verkürzt.

Regeln zur Vereinheitlichung der Datenquellen: Die Datenquellen werden einzeln mit je einer Xcerpt-Regel in eine einheitliche, allerdings auch heterogene Struktur überführt. Das Element `entry` steht für jedes Buch oder jeden Artikel und hat als Kindelemente Titel und Autoren. Daten, wie z.B. Jahresangaben, Preise und Dokumentlinks werden innerhalb von speziellen Elementen angeordnet, um die Abhängigkeit zu ihren jeweiligen Quelldateien beizubehalten (`amazoninfo`, `pmsinfo` und `dblpinfo`). Dies geschieht aus Gründen der Einfachheit, da in diesem Szenario nur Redundanzen über Buchtitel und Autoren in einer einfachen Weise beseitigt werden. Aus dem selben Grund werden einige Einträge aus den Quelldokumenten nicht in das neue Dokument mit aufgenommen.

```
1 CONSTRUCT
2 science [
3   all entry [
4     var Title,
5     authors [
6       all var Author
```

[1] http://www.informatik.uni-trier.de/~ley/db/index.html
[2] http://www.amazon.de
[3] http://www.pms.informatik.uni-muenchen.de/publikationen/index-alt.html

```
    ],
    dblpinfo {
      category { "article" },
      date {
        var Year
      },
    }
  ]
]
FROM
in {
  resource {"file:dblp.xml","xml"},
  article {{
    var Author -> author {{}},
    var Title -> title {{}},
    var year -> year {{}}
  }}
END

CONSTRUCT
  science [
    all entry [
      title [ var Title ],
      authors [
        all var Author
      ],
      amazoninfo [
        category { var Category },
        var Url,
        price { var Price },
        date {
          optional year { var Year },
          optional month { var Month }
        }
      ]
    ]
  ]
FROM
in {
  resource {"file:amazon.xml","xml"},
  productinfo {{
    details {{
      attributes {
        var Url -> url {{}}
      },
      productname { var Title },
      catalog { var Category },
      authors [[
        var Author -> author {{}}
      ]],
      optional releasedate { /(var month -> [A-Z][a-zA-Z]*) (var Year -> [12][09][09][09])/ },
      ourprice { var AmazonPrice }
    }}
  }}
}
END

CONSTRUCT
  science [
    all entry [
      title [ var Title ],
      authors [
        all var Author,
```

```
70          var LastAuthor
71        ],
72        pmsinfo {
73          documents [
74            all document {
75              description { var Description },
76              link {
77                var Document
78              }
79            }
80          ],
81          date {
82            year [ var Year ]
83          }
84        }
85      ]
86    ]
87  FROM
88  in {
89    resource {"file:pms.html", "html"},
90    html [[
91      h3 { var Year },
92      ul [[
93        br {},
94        /((var Author -> [A-Z][a-zA-Z]*), )*((var LastAuthor -> [A-Z][a-zA-Z]*):)/,
95        br {},
96        em { var Title },
97        a {
98          attributes {
99            href { var Document }
100         },
101         var Description
102       }
103     ]]
104   ]]
105 END
```

Regel zur Zusammenführung der Datenquellen: Diese Regel wird auf die Ergebnisse der drei oberen Regel angewandt und führt einen einfachen Join über die Buchtitel und die Autoren durch. Sind diese Werte zweier `entry`-Terme gleich, werden sie als die selben Bücher oder Artikel angesehen. Alle weiteren Zusatzinformationen werden einfach hinzugefügt. Man könnte an dieser Stelle auch den Referenzmechanismus von Xcerpt benutzen, wie er in der zweiten Anfrage von Abschnitt 3.1.8.1 verwendet wird. Dies würde sich besonders anbieten, wenn man die Daten auf der Mediatorenschicht als Xcerpt-Terme abspeichern möchte und nicht im XML-Format.

```
1   CONSTRUCT
2     science [
3       all entry {
4         var Title,
5         var Authors,
6         optional var Amazoninfo,
7         optional var Dblpinfo,
8         optional var Pmsinfo
9       }
10    ]
11  FROM
12    science [[
```

```
    entry {{
      var Title -> title {{}},
      var Authors -> authors {{}},
      optional var Amazoninfo -> amazoninfo {{}},
      optional var Dblpinfo -> dblpinfo {{}},
      optional var Pmsinfo -> pmsinfo {{}}
    }},
    entry {{
      var Title -> title {{}},
      var Authors -> authors {{}},
      optional var Amazoninfo -> amazoninfo {{}},
      optional var Dblpinfo -> dblpinfo {{}},
      optional var Pmsinfo -> pmsinfo {{}}
    }}
  ]]
END
```

Anfrageregel:

Eine benutzerspezifische Anfrage, die an den Mediator gestellt wird, könnte nun lauten: "Ich habe von einem Buch oder Artikel über visuelle Anfragesprachen gehört. Einer der Autoren heißt "Bry". Gib mir alle Informationen zu diesem Buch, wie z.B. Preise, wenn es käuflich ist oder Lesebeispiele. Sollte es nicht käuflich sein, möchte ich Informationen zu einem Buch des Autoren "Bry", welches käuflich zu erwerben ist."

Diese Anfrage kann dann folgendermaßen gestellt werden, wobei angenommen wird, dass die Daten des Mediators in der Datei "mediator_data.xcerpt" abgespeichert sind.

```
CONSTRUCT
  result {
    all entry {
      title { var Title },
      var Authors,
      optional documents {
        all document { var Document }
      },
      optional var Order
    },
    optional some 1 otherBooks {
      var OtherEntry
    }
  }
FROM
  in {
    resource {"file:mediator_data.xcerpt","xcerpt"},
    science {{
      var Entry -> entry {{
        title {var Title},
        var Authors -> authors {{
          desc /.*Bry.*/
        }},
        optional pmsinfo {{
          documents { var Document }
        }},
        IF
          without amazoninfo {
            price {var NoPrice}
          }
        THEN
```

```
          in {
            resource {"file:mediator_data.xcerpt","xcerpt"},
            optional var otherEntry -> entry {
              authors { desc /.*Bry.*/ },
              amazoninfo {{}}
            }
          }
        ELSE
          var Order -> amazoninfo {{}}
      }} where { and { var Title eq /.*visual.*/, var Title eq /.*querying.*/ } }
    }}
  }
END
```

Das Ergebnis: Die Anfrage liefert dann folgenden Datenterm als Ergebnis zurück.

```
result {
  entry {
    title { "A Visual Language for Web Querying and Reasoning" },
    authors {
      author { "Sacha Berger" },
      author { "François Bry" },
      author { "Sebastian Schaffert" }
    },
    documents {
      document {
        description { "Abstract (.txt)" },
        link { "PMS-FB/PMS-FB-2003-6.abstract" }
      },
      document {
        description { "complete text (.pdf, 320K)" },
        link { "PMS-FB/PMS-FB-2003-6.pdf" }
      }
    }
  },
  entry {
    title { "Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of
            XML and Semistructured Data" },
    authors {
      author { "Sacha Berger" },
      author { "François Bry" },
      author { "Sebastian Schaffert" },
      author { "Christoph Wieser" }
    },
    documents {
      document {
        description { "complete text (.pdf, 180K)" },
        link { "PMS-FB/PMS-FB-2003-2.pdf" }
      }
    }
  },
  otherBooks {
    entry {
      title { "Deductive and Object-Oriented Databases" },
      authors {
        author { "François Bry" },
        author { "Raghu Ramakrishnan" },
        author { "Kotagiri Ramamohanarao" }
      },
      amazoninfo [
        category { "Book" },
```

```
46          url { "http://www.amazon.de/exec/obidos/ASIN/3540637923" },
47          price { "EUR 55,64" },
48          date {
49            year { 1997},
50            month { "December" }
51          }
52        }
53      }
54    }
55  }
```

Kapitel 4

Ausblick

Es wurde in dieser Arbeit gezeigt, wie Anfragen auf semistrukturierte Daten mit Xcerpt durchgeführt werden können und für welche Anfragearten Xcerpt durch seinen Patterngedanken geeignet ist. Weiterhin wurde ersichtlich, wie eine logikbasierte Anfragesprache arbeitet und weshalb gerade eine logische Anfragesprache für die Suche im Web große Vorteile bietet.

Allerdings ist Xcerpt noch in seiner Entwicklung und es müssen noch viele Aspekte für diese Sprache untersucht und eingebettet werden. So ist ein umfassendes Typsystem zu integrieren, welches den Umgang mit Schemata ermöglicht, vor allem was XML Schema und DTD betrifft. Weiterhin sollte eine Datenbankanfragesprache Updateoperationen leisten können, um Anwendungen ein dynamischeres Verhalten zu ermöglichen. Vor allem aber muss im Bereich der Optimierung und der Effizienzsteigerung Forschungsarbeit geleistet werden, damit diese Sprache sich auch im kommerziellen Bereich etablieren kann. In all diesen Forschungsfelder werden in der nächsten Zeit am Lehrstuhl für Programmier- und Modellierungssprachen der LMU München und in anderen Forschungseinrichtungen Europas intensive Arbeit geleistet werden, damit das Web nicht eine Sammlung zusammenhangloser Informationen bleibt.

Kapitel 5

Anhang

5.1 Dokumente der “XML Query Use Cases” vom X3C

5.1.1 Use Case “XMP: Experiences and Exemplars”

5.1.1.1 Quelldateien

Datei “bib.xml”:

```
<bib>
  <book year="1994">
    <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">
    <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">
    <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">
    <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>
```

Datei “books.xml”:

```
<chapter>
  <title>Data Model</title>

  <section>
    <title>Syntax For Data Model</title>
  </section>

  <section>
    <title>XML</title>

    <section>
      <title>Basic Syntax</title>
    </section>

    <section>
      <title>XML and Semistructured Data</title>
    </section>
  </section>
</chapter>
```

Datei “prices.xml”:

```
<prices>
  <book>
    <title>Advanced Programming in the Unix environment</title>
    <source>www.amazon.com</source>
    <price>65.95</price>
  </book>

  <book>
    <title>Advanced Programming in the Unix environment</title>
    <source>www.bn.com</source>
    <price>65.95</price>
  </book>

  <book>
    <title>TCP/IP Illustrated</title>
    <source>www.amazon.com</source>
    <price>65.95</price>
  </book>
```

```
19
20    <book>
21      <title>TCP/IP Illustrated</title>
22      <source>www.bn.com</source>
23      <price>65.95</price>
24    </book>
25
26    <book>
27      <title>Data on the Web</title>
28      <source>www.amazon.com</source>
29      <price>34.95</price>
30    </book>
31
32    <book>
33      <title>Data on the Web</title>
34      <source>www.bn.com</source>
35      <price>39.95</price>
36    </book>
37  </prices>
```

Datei “reviews.xml”:

```
1   <reviews>
2     <entry>
3       <title>Data on the Web</title>
4       <price>34.95</price>
5       <review>A very good discussion of semi-structured database
6       systems and XML.</review>
7     </entry>
8     <entry>
9       <title>Advanced Programming in the Unix environment</title>
10      <price>65.95</price>
11      <review>A clear and detailed discussion of UNIX
12      programming.</review>
13    </entry>
14    <entry>
15      <title>TCP/IP Illustrated</title>
16      <price>65.95</price>
17      <review>One of the best books on TCP/IP.</review>
18    </entry>
19  </reviews>
```

5.1.1.2 Anfragen

Q1: *List books published by Addison-Wesley after 1991, including their year and title.*

Lösung mit XQuery:

```
1   <bib>
2    {
3     for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
4     where $b/publisher = "Addison-Wesley" and $b/@year > 1991
5     return
6       <book year="{ $b/@year }">
7        { $b/title }
8       </book>
```

```
 }
</bib>
```

Lösung mit Xcerpt:

```
GOAL
out {
  resource { "stdout:", "html" },
  var C
}
FROM
var C -> bib {{}}
END

CONSTRUCT
bib [
  all book [
    attributes  {
      year [ var Year ]
    },
    var Title
  ]
]
FROM
in {
  resource [ "file:bib.xml", "xml" ],
  bib [[
    book [[
      attributes  {{
        year { var Year::integer }
      }},
      var Title -> title {{}},
      publisher { "Addison-Wesley" }
    ]]
  ]]
} where { var Year::integer gt 1991 }
END
```

Q2: *Create a flat list of all the title-author pairs, with each pair enclosed in a "result" element.*

Lösung mit XQuery:

```
<results>
  {
    for $b in doc("http://bstore1.example.com/bib.xml")/bib/book,
        $t in $b/title,
        $a in $b/author
    return
        <result>
            { $t }
            { $a }
        </result>
  }
</results>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all result [
    var Title,
    var Author
  ]
]
FROM
in {
  resource { "file:xmp-bib.xml", "xml" },
  bib [[
    book [[
      var Title -> title {{}},
      var Author -> author {{}}
    ]]
  ]]
}
END
```

Q3: *For each book in the bibliography, list the title and authors, grouped inside a ”result” element.*

Lösung mit XQuery:

```
<results>
{
    for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
    return
        <result>
            { $b/title }
            { $b/author  }
        </result>
}
</results>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all result [
    var Title,
    all var Author
  ]
]
FROM
in {
  resource { "file:xmp-bib.xml", "xml" },
  bib [[
    book [[
      var Title -> title {{}},
      var Author -> author {{}}
    ]]
  ]]
}
END
```

Q4: *For each author in the bibliography, list the author's name and the titles of all books by that author, grouped inside a "result" element.*

Lösung mit XQuery:

```
<results>
  {
    let $a := doc("http://bstore1.example.com/bib/bib.xml")//author
    for $last in distinct-values($a/last),
        $first in distinct-values($a[last=$last]/first)
    order by $last, $first
    return
        <result>
            <author>
               <last>{ $last }</last>
               <first>{ $first }</first>
            </author>
            {
                for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
                where some $ba in $b/author
                      satisfies ($ba/last = $last and $ba/first=$first)
                return $b/title
            }
        </result>
  }
</results>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all result [
    var Author,
    all var Title
  ]
]
FROM
in {
  resource { "file:xmp-bib.xml", "xml" },
  bib [[
    book [[
      var Title -> title {{}},
      var Author -> author {{}}
    ]]
  ]]
}
END
```

Q5: *For each book found at both bstore1.example.com and bstore2.example.com, list the title of the book and its price from each source.*

Lösung mit XQuery:

```
<books-with-prices>
  {
    for $b in doc("http://bstore1.example.com/bib.xml")//book,
        $a in doc("http://bstore2.example.com/reviews.xml")//entry
```

```
    where $b/title = $a/title
    return
        <book-with-prices>
            { $b/title }
            <price-bstore2>{ $a/price/text() }</price-bstore2>
            <price-bstore1>{ $b/price/text() }</price-bstore1>
        </book-with-prices>
  }
</books-with-prices>
```

Lösung mit Xcerpt:

```
CONSTRUCT
books-with-prices [
  all book-with-prices [
    title [
      var T
    ],
    price-amazon [ var Pa ],
    price-bn [ var Pb ]
  ]
]
FROM
and {
  in {
    resource { "file:xmp-bib.xml", "xml" },
    bib [[
      book [[
        title [ var T ],
        price [ var Pa ]
      ]]
    ]]
  },
  in {
    resource { "file:xmp-reviews.xml", "xml" },
    reviews [[
      entry [[
        title [ var T ],
        price [ var Pb ]
      ]]
    ]]
  }
}
END
```

Q6: *For each book that has at least one author, list the title and first two authors, and an empty "et-al" element if the book has additional authors.*

Lösung mit XQuery:

```
<bib>
  {
    for $b in doc("http://bstore1.example.com/bib.xml")//book
    where count($b/author) > 0
    return
        <book>
            { $b/title }
            {
```

```
                for $a in $b/author[position()<=2]
                return $a
            }
            {
                if (count($b/author) > 2)
                 then <et-al/>
                 else ()
            }
        </book>
  }
</bib>
```

Lösung mit Xcerpt:

```
GOAL
  result { all var }
FROM
  var C -> book {{}}
END

CONSTRUCT
  book {
    var Title,
    first 2 var Author,
    et_al []
  }
FROM
  books_with_authorcount {{
    books {{
      book {{
        var Title -> title {{}},
        var Author -> author {{}}
      }},
      authorcount { var Count::integer }
    }}
  }}
  where { var Count::integer geq 3 }
END

CONSTRUCT
  book {
    var Title,
    all var Author
  }
FROM
  books_with_authorcount {{
    books {{
      book {{
        var Title -> title {{}},
        var Author -> author {{}}
      }},
      authorcount { var Count::integer }
    }}
  }}
  where { var Count::integer leq 2 }
END

CONSTRUCT
  books_with_authorcount [
    all books [
      var Book,
```

```
48          authorcount {
49            count ( all var Author )
50          }
51        ]
52    ]
53  FROM
54    in {
55      resource { "file:xmp-bib.xml", "xml" },
56      bib [[
57        var Book -> book {{
58            var Author -> author {{}}
59        }}
60      ]]
61    }
62  END
```

Q7: *List the titles and years of all books published by Addison-Wesley after 1991, in alphabetic order.*

Lösung mit XQuery:

```
1  <bib>
2    {
3      for $b in doc("http://bstore1.example.com/bib.xml")//book
4      where $b/publisher = "Addison-Wesley" and $b/@year > 1991
5      order by $b/title
6      return
7          <book>
8              { $b/@year }
9              { $b/title }
10         </book>
11   }
12 </bib>
```

Lösung mit Xcerpt:

```
1  CONSTRUCT
2  bib [
3    all book [
4      attributes  {
5        year { var Year::integer }
6      },
7      title { var Title }
8    ] ordering [Title] literal
9  ]
10 FROM
11 in {
12   resource { "file:xmp-bib.xml", "xml" },
13   bib [[
14     var Book -> book {{
15       attributes  {
16         year { var Year::integer }
17       },
18       title { var Title },
19       publisher { "Addison-Wesley" }
20     }}
21   ]]
22 } where { var Year::integer gt 1991 }
23 END
```

Q8: *Find books in which the name of some element ends with the string "or" and the same element contains the string "Suciu" somewhere in its content. For each such book, return the title and the qualifying element.*

Lösung mit XQuery:

```
for $b in doc("http://bstore1.example.com/bib.xml")//book
let $e := $b/*[contains(string(.), "Suciu")
               and ends-with(local-name(.), "or")]
where exists($e)
return
    <book>
        { $b/title }
        { $e }
    </book>
```

Lösung mit Xcerpt:

```
CONSTRUCT
bib [
  all book [
    var Title,
    var X
  ]
]
FROM
in {
  resource { "file:xmp-bib.xml", "xml" },
  bib [[
    book {{
      var Title -> title {{}},
      desc var X -> /.*or/ {{
        desc /.*Suciu.*/
      }}
    }}
  ]]
}
END
```

Q9: *In the document "books.xml", find all section or chapter titles that contain the word "XML", regardless of the level of nesting.*

Lösung mit XQuery:

```
<results>
  {
    for $t in doc("books.xml")//(chapter | section)/title
    where contains($t/text(), "XML")
    return $t
  }
</results>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all var Title
]
FROM
in {
  resource { "file:xmp-books.xml", "xml" },
  or {
    desc chapter [[
      var Title -> title [[ /.*XML.*/ ]]
    ]],
    desc section [[
      var Title -> title [[ /.*XML.*/ ]]
    ]]
  }
}
END
```

Q10: *In the document ”prices.xml”, find the minimum price for each book, in the form of a ”minprice” element with the book title as its title attribute.*

Lösung mit XQuery:

```
<results>
  {
    let $doc := doc("prices.xml")
    for $t in distinct-values($doc//book/title)
    let $p := $doc//book[title = $t]/price
    return
      <minprice title="{ $t }">
        <price>{ min($p) }</price>
      </minprice>
  }
</results>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all minprice [
    price {
      attributes  {
        title { var T }
      },
      min ( all var Price::float )
    }
  ]
]
FROM
in {
  resource { "file:xmp-prices.xml", "xml" },
  prices [[
    book [[
      title [ var T ],
      price [ var Price::float ]
```

```
    ]]
  ]]
}
END
```

Q11: *For each book with an author, return the book with its title and authors. For each book with an editor, return a reference with the book title and the editor's affiliation.*

Lösung mit XQuery:

```
<bib>
{
        for $b in doc("http://bstore1.example.com/bib.xml")//book[author]
        return
            <book>
                { $b/title }
                { $b/author }
            </book>
}
{
        for $b in doc("http://bstore1.example.com/bib.xml")//book[editor]
        return
          <reference>
            { $b/title }
            {$b/editor/affiliation}
          </reference>
}
</bib>
```

Lösung mit Xcerpt:

```
GOAL
  out {
    resource { "stdout:", "html" },
    results [ all var Book  ]
  }
FROM
  or {
    var Book -> book {{}},
    var Book -> reference {{}}
  }
END

CONSTRUCT
  book [
    var Title,
    all var Author
  ]
FROM
  in {
    resource { "file:xmp-bib.xml", "xml" },
    bib [[
      book [[
        var Title -> title {{}},
        var Author -> author {{}}
      ]]
    ]]
  }
```

```
END

CONSTRUCT
  reference [
    var Title,
    affiliation [ var Affiliation ]
  ]
FROM
  in {
    resource { "file:xmp-bib.xml", "xml" },
    bib [[
      book [[
        var Title -> title {{}},
        editor [[
          affiliation [[ var Affiliation ]]
        ]]
      ]]
    ]]
  }
END
```

Q12: *Find pairs of books that have different titles but the same set of authors (possibly in a different order).*

Lösung mit XQuery:

```
<bib>
{
    for $book1 in doc("http://bstore1.example.com/bib.xml")//book,
        $book2 in doc("http://bstore1.example.com/bib.xml")//book
    let $aut1 := for $a in $book1/author
                 order by $a/last, $a/first
                 return $a
    let $aut2 := for $a in $book2/author
                 order by $a/last, $a/first
                 return $a
    where $book1 << $book2
    and not($book1/title = $book2/title)
    and deep-equal($aut1, $aut2)
    return
        <book-pair>
            { $book1/title }
            { $book2/title }
        </book-pair>
}
</bib>
```

Lösung mit Xcerpt:

```
CONSTRUCT
bib {
  all titlebib {
    title { var Title1 },
    title { var Title2 }
  }
}
FROM
temp [[
```

```
10    title_authorset {{
11      title { var Title1},
12      var Author_set -> author_set {{}}
13    }},
14    title_authorset {{
15      title { var Title2},
16      var Author_set -> author_set {{}}
17    }}
18  ]] where { var Title1 ne Title2 }
19
20  CONSTRUCT
21  temp {
22    all title_authorset {
23      title { var Title },
24      author_set { all var Author }
25    }
26  }
27  FROM
28  in {
29    resource { "file:xmp-bib.xml", "xml" },
30    bib [[
31      book {{
32        title { var Title },
33        var Author -> author {{}}
34      }}
35    ]]
36  }
37  END
```

5.1.2 Use Case “TREE: Queries That Preserve Hierarchy”

5.1.2.1 Quelldatei

Datei “book.xml”:

```
<?xml version="1.0"?>
<!DOCTYPE book SYSTEM "book.dtd">
<book>
  <title>Data on the Web</title>
  <author>Serge Abiteboul</author>
  <author>Peter Buneman</author>
  <author>Dan Suciu</author>
  <section id="intro" difficulty="easy" >
    <title>Introduction</title>
    <p>Text ... </p>
    <section>
      <title>Audience</title>
      <p>Text ... </p>
    </section>
    <section>
      <title>Web Data and the Two Cultures</title>
      <p>Text ... </p>
      <figure height="400" width="400">
        <title>Traditional client/server architecture</title>
        <image source="csarch.gif"/>
      </figure>
      <p>Text ... </p>
    </section>
  </section>
  <section id="syntax" difficulty="medium" >
    <title>A Syntax For Data</title>
    <p>Text ... </p>
    <figure height="200" width="500">
      <title>Graph representations of structures</title>
      <image source="graphs.gif"/>
    </figure>
    <p>Text ... </p>
    <section>
      <title>Base Types</title>
      <p>Text ... </p>
    </section>
    <section>
      <title>Representing Relational Databases</title>
      <p>Text ... </p>
      <figure height="250" width="400">
        <title>Examples of Relations</title>
        <image source="relations.gif"/>
      </figure>
    </section>
    <section>
      <title>Representing Object Databases</title>
      <p>Text ... </p>
    </section>
  </section>
</book>
```

5.1.2.2 Anfragen

Q1: *Prepare a (nested) table of contents for Book1, listing all the sections and their titles. Preserve the original attributes of each <section> element, if any.*

Lösung mit XQuery:

```
declare function local:toc($book-or-section as element()) as element()*
{
    for $section in $book-or-section/section
    return
      <section>
         { $section/@* , $section/title , local:toc($section) }
      </section>
};

<toc>
   {
     for $s in doc("book.xml")/book return local:toc($s)
   }
</toc>
```

Lösung mit Xcerpt:

```
GOAL
var Result
FROM
and {
  in {
    resource { "file:book-tree.xml", "xml" },
    var Root -> book {{}}
  },
  transform [ var Root, var Result ]
}
END

CONSTRUCT
  transform [
    var In ,
    section [
      optional var Attributes
      titel { var Titel },
      all var NewSubterm
    ]
  ]
FROM
  and {
    section {{
      optional var Attributes -> attributes {{}}
      title { var Titel },
      desc var Rest -> section {{}}
    }} :< var In,
    transform [ var Rest, var NewSubterm ]
  }
END

CONSTRUCT
  transform [
    var In,
```

```
    section [
      optional var Attributes
      titel { var Titel }
    ]
  ]
FROM
  section {{
    optional var Attributes -> attributes {{}}
    title { var Titel },
    without desc var Rest -> section {{}}
  }} :< var In
END

CONSTRUCT
  transform [
    var In,
    result [ all var NewSubterm ]
  ]
FROM
  and {
    var Label {{
      var Subterm
    }} where { var Label != "section" }
    :< var In,
    transform [ var Subterm, var NewSubterm ]
  }
END
```

Q2: *Prepare a (flat) figure list for Book1, listing all the figures and their titles. Preserve the original attributes of each “figure” element, if any.*

Lösung mit XQuery:

```
<figlist>
  {
    for $f in doc("book.xml")//figure
    return
        <figure>
            { $f/@* }
            { $f/title }
        </figure>
  }
</figlist>
```

Lösung mit Xcerpt:

```
CONSTRUCT
figlist [
  all figure {
    attributes  { all optional var Attributes },
    var Title
  }
]
FROM
in {
  resource { "file:book-tree.xml", "xml" },
  desc figure {{
    attributes  { optional var Attributes },
```

```
    var Title -> title {{}}
  }}
}
END
```

Q3: *How many sections are in Book1, and how many figures?*

Lösung mit XQuery:

```
<section_count>{ count(doc("book.xml")//section) }</section_count>,
<figure_count>{ count(doc("book.xml")//figure) }</figure_count>
```

Lösung mit Xcerpt:

```
CONSTRUCT
count [
  section_count {
    count ( all optional var Section )
  },
  figure_count {
    count ( all optional var Figure )
  }
]
FROM
in {
  resource { "file:book-tree.xml", "xml" },
  and {
    desc optional var Section -> section {{}},
    desc optional var Figure -> figure {{}}
    }
  }
END
```

Q4: *How many top-level sections are in Book1?*

Lösung mit XQuery:

```
<top_section_count>
 {
   count(doc("book.xml")/book/section)
 }
</top_section_count>
```

Lösung mit Xcerpt:

```
CONSTRUCT
top_section_count [
  count ( all optional var Topsection )
]
FROM
in {
  resource { "file:book-tree.xml", "xml" },
  book {{
```

```
    optional var Topsection -> section {{}}
  }}
}
END
```

Q5: *Make a flat list of the section elements in Book1. In place of its original attributes, each section element should have two attributes, containing the title of the section and the number of figures immediately contained in the section.*

Lösung mit XQuery:

```
<section_list>
  {
    for $s in doc("book.xml")//section
    let $f := $s/figure
    return
        <section title="{ $s/title/text() }" figcount="{ count($f) }"/>
  }
</section_list>
```

Lösung mit Xcerpt:

```
CONSTRUCT
section_list {
  all section {
    attributes  {
      title { var Title },
      figcount {
        count ( all optional var Figure )
      }
    }
  }
}
FROM
in {
  resource { "file:book-tree.xml", "xml" },
  desc section {{
    title { var Title },
    optional var Figure -> figure {{}}
  }}
}
END
```

Q6: *Make a nested list of the section elements in Book1, preserving their original attributes and hierarchy. Inside each section element, include the title of the section and an element that includes the number of figures immediately contained in the section.*

Lösung mit XQuery:

```
declare function local:section-summary($book-or-section as element())
  as element()*
{
  for $section in $book-or-section
  return
```

```
6       <section>
7          { $section/@* }
8          { $section/title }
9          <figcount>
10           { count($section/figure) }
11         </figcount>
12         { local:section-summary($section/section) }
13      </section>
14  };
15
16  <toc>
17    {
18      for $s in doc("book.xml")/book/section
19      return local:section-summary($s)
20    }
21  </toc>
```

Lösung mit Xcerpt:

```
1   GOAL
2   var Result
3   FROM
4   and {
5     in {
6       resource { "file:book-tree.xml", "xml" },
7       var Root -> book {{}}
8     },
9     alter [
10      var Root,
11      var Result
12    ]
13  }
14  END
15
16  CONSTRUCT
17  alter [
18    var In,
19    section [
20      optional var Attributes,
21      titel { var Titel },
22      figcount {
23        count (  all optional var Fig )
24      },
25      all var NewSubterm
26    ]
27  ]
28  FROM
29  and {
30    section {{
31      optional var Attributes -> attributes {{}},
32      title { var Titel },
33      optional var Fig -> figure {{}},
34      var Rest
35    }} :< var In,
36    alter [
37      var Rest,
38      var NewSubterm
39    ]
40  }
41  END
42
```

```
43 CONSTRUCT
44 alter [
45   var In,
46   section [
47     optional var Attributes,
48     titel { var Title },
49     figcount {
50       count ( all optional var Fig )
51     }
52   ]
53 ]
54 FROM
55 section {{
56   optional var Attributes -> attributes {{}},
57   title { var Title },
58   optional var Fig -> figure {{}},
59   without desc section {{}}
60 }} :< var In
61 END
62
63 CONSTRUCT
64 alter [
65   var In,
66   result [
67     all var NewSubterm
68   ]
69 ]
70 FROM
71 and {
72   var OrigLabel {{
73     var OrigSubterm
74   }} where { var Label != "section" }
75   :< var In,
76   alter [
77     var OrigSubterm,
78     var NewSubterm
79   ]
80 }
81 END
```

5.1.3 Use Case "SEQ: Queries Based on Sequences"

5.1.3.1 Quelldatei

Datei "report1.xml":

```
<report>
  <section>
    <section_title>Procedure</section_title>
     <section_content>
      The patient was taken to the operating room where she was placed
      in supine position and
      <anesthesia>induced under general anesthesia_</anesthesia>
      <prep>
        <action>A Foley catheter was placed to decompress the bladder</action>
        and the abdomen was then prepped and draped in sterile fashion.
      </prep>
      <incision>
        A curvilinear incision was made
        <geography>in the midline immediately infraumbilical</geography>
        and the subcutaneous tissue was divided
        <instrument>using electrocautery.</instrument>
      </incision>
      The fascia was identified and
      <action>#2 0 Maxon stay sutures were placed on each side of the midline.
      </action>
      <incision>
        The fascia was divided using
        <instrument>electrocautery</instrument>
        and the peritoneum was entered.
      </incision>
      <observation>The small bowel was identified.</observation>
      and
      <action>
        the
        <instrument>Hasson trocar</instrument>
        was placed under direct visualization.
      </action>
      <action>
        The
        <instrument>trocar</instrument>
        was secured to the fascia using the stay sutures.
      </action>
     </section_content>
  </section>
</report>
```

5.1.3.2 Anfragen

Q1: *In the Procedure section of Report1, what Instruments were used in the second Incision?*

Lösung mit XQuery:

```
for $s in doc("report1.xml")//section[section.title = "Procedure"]
return ($s//incision)[2]/instrument
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all var Instrument
}
FROM
temp {{
  desc var Instrument -> instrument {{}}
}}
END

CONSTRUCT
temp [
  first ( drop 1 ( all var Incision ) )
]
FROM
in {
  resource { "file:report1.xml", "xml" },
  desc section [[
    section.title { "Procedure" },
    desc var Incision -> incision {{}}
  ]]
}
END
```

Q2: *In the Procedure section of Report1, what are the first two Instruments to be used?*

Lösung mit XQuery:

```
for $s in doc("report1.xml")//section[section.title = "Procedure"]
return ($s//instrument)[position()<=2]
```

Lösung mit Xcerpt:

```
CONSTRUCT
result [
  first 2 ( all var Instrument )
]
FROM
in {
  resource { "file:report1.xml", "xml" },
  desc section [[
    section_title { "Procedure" },
    desc var Instrument -> instrument {{}}
  ]]
}
END
```

Q3: *In Report1, what Instruments were used in the first two Actions after the second Incision?*

Lösung mit XQuery:

```
let $i2 := (doc("report1.xml")//incision)[2]
for $a in (doc("report1.xml")//action)[. >> $i2][position()<=2]
return $a//instrument
```

1. Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all var Instrument
}
FROM
temp {{
  action {{
    desc var Instrument -> instrument {{}}
  }}
}}
END

CONSTRUCT
temp [
  first 2 ( all var Action )
]
FROM
in {
  resource { "file:report1.xml", "xml" },
  desc var X [[
    incision {{}},
    incision {{}},
    desc var Action -> action {{}}
  ]]
}
END
```

2. Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all var Instrument
}
FROM
temp2 {{
  action {{
    desc var Instrument -> instrument {{}}
  }}
}}
END

CONSTRUCT
temp2 {
  first 2 ( action {all var Action})
}
FROM
temp1 [
```

```
  position 2 I_A [[
    var Action -> actions {{}}
  ]]
]

CONSTRUCT
temp1 [
  all I_A [
    var Inc,
    all actions { optional var Action }
  ]
FROM
in {
  resource { "file:report1.xml", "xml" },
  desc var X [[
    var Inc -> incision {{}},
    optional desc var Action -> action {{}}
  ]]
}
END
```

Q4: *In Report1, find Procedureßections where no Anesthesia element occurs before the first Incision*

Lösung mit XQuery:

```
for $p in doc("report1.xml")//section[section.title = "Procedure"]
where not(some $a in $p//anesthesia satisfies
        $a << ($p//incision)[1] )
return $p
```

Lösung mit Xcerpt:

```
CONSTRUCT
  result {
    all var Section
 }
FROM
  temp [
    section_and_first_Inc [[
      var Section -> section {{}},
      content [[
        without anesthesia {{}},
        incision {{}}
      ]]
    ]]
  ]
END

CONSTRUCT
  temp [
    all section_and_first_Inc [
      var Section,
      first (
        all content [
          all optional var X,
          var Inc
        ]
```

```
      )
    ]
  ]
FROM
  in {
    resource { "file:report1.xml", "xml" },
    desc var Section -> section {{
      section.title { "Procedure" },
      section.content [[
        optional var X,
        var Inc -> incision {{}}
      ]]
    }}
  }
END
```

Q5: *In Report1, what happened between the first Incision and the second Incision?*

Lösung mit XQuery:

```
<critical_sequence>
 {
  let $proc := doc("report1.xml")//section[section.title="Procedure"][1],
      $i1 :=  ($proc//incision)[1],
      $i2 :=  ($proc//incision)[2]
  for $n in $proc//node() except $i1//node()
  where $n >> $i1 and $n << $i2
  return $n
 }
</critical_sequence>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result [
  all var Action
]
FROM
in {
  resource { "file:report1.xml", "xml" },
  desc var X [[
    optional var Y {{}},
    incision {{}},
    var Action,
    incision {{}}
  ]] where { var Y != "incision" }
}
END
```

5.1.4 Use Case “R: Access to Relational Data”

5.1.4.1 Quelldateien

Datei “users.xml”:

```
  <users>
  <user_tuple>
    <userid>U01</userid>
    <name>Tom Jones</name>
    <rating>B</rating>
  </user_tuple>
 <user_tuple>
    <userid>U02</userid>
    <name>Mary Doe</name>
    <rating>A</rating>
  </user_tuple>
 <user_tuple>
    <userid>U03</userid>
    <name>Dee Linquent</name>
    <rating>D</rating>
  </user_tuple>
 <user_tuple>
    <userid>U04</userid>
    <name>Roger Smith</name>
    <rating>C</rating>
  </user_tuple>
 <user_tuple>
    <userid>U05</userid>
    <name>Jack Sprat</name>
    <rating>B</rating>
  </user_tuple>
 <user_tuple>
    <userid>U06</userid>
    <name>Rip Van Winkle</name>
    <rating> B</rating>
  </user_tuple>
</users>
```

Datei “bids.xml”:

```
 <bids>
  <bid_tuple>
    <userid>U02</userid>
    <itemno>1001</itemno>
    <bid>35</bid>
    <bid_date>1999-01-07</bid_date>
    </bid_tuple>

 <bid_tuple>
    <userid>U04</userid>
    <itemno>1001</itemno>
    <bid>40</bid>
    <bid_date>1999-01-08</bid_date>
    </bid_tuple>

 <bid_tuple>
    <userid>U02</userid>
```

```
18      <itemno>1001</itemno>
19      <bid>45</bid>
20      <bid_date>1999-01-11</bid_date>
21      </bid_tuple>
22
23    <bid_tuple>
24      <userid>U04</userid>
25      <itemno>1001</itemno>
26      <bid>50</bid>
27      <bid_date>1999-01-13</bid_date>
28      </bid_tuple>
29
30    <bid_tuple>
31      <userid>U02</userid>
32      <itemno>1001</itemno>
33      <bid>55</bid>
34      <bid_date>1999-01-15</bid_date>
35      </bid_tuple>
36
37    <bid_tuple>
38      <userid>U01</userid>
39      <itemno>1002</itemno>
40      <bid>400</bid>
41      <bid_date>1999-02-14</bid_date>
42      </bid_tuple>
43
44
45    <bid_tuple>
46      <userid>U02</userid>
47      <itemno>1002</itemno>
48      <bid>600</bid>
49      <bid_date>1999-02-16</bid_date>
50      </bid_tuple>
51
52    <bid_tuple>
53      <userid>U03</userid>
54      <itemno>1002</itemno>
55      <bid>800</bid>
56      <bid_date> 1999-02-17</bid_date>
57      </bid_tuple>
58
59    <bid_tuple>
60      <userid>U04</userid>
61      <itemno>1002</itemno>
62      <bid>1000</bid>
63      <bid_date>1999-02-25</bid_date>
64      </bid_tuple>
65
66    <bid_tuple>
67      <userid>U02</userid>
68      <itemno>1002</itemno>
69      <bid>1200</bid>
70      <bid_date>1999-03-02</bid_date>
71      </bid_tuple>
72
73    <bid_tuple>
74      <userid>U04</userid>
75      <itemno>1003</itemno>
76      <bid>15</bid>
77      <bid_date>1999-01-22</bid_date>
78      </bid_tuple>
79
80    <bid_tuple>
```

```
81      <userid>U05</userid>
82      <itemno>1003</itemno>
83      <bid>20</bid>
84      <bid_date>1999-02-03</bid_date>
85      </bid_tuple>
86
87    <bid_tuple>
88      <userid>U01</userid>
89      <itemno>1004</itemno>
90      <bid>40</bid>
91      <bid_date>1999-03-05</bid_date>
92      </bid_tuple>
93
94    <bid_tuple>
95      <userid>U03</userid>
96      <itemno>1007</itemno>
97      <bid>175</bid>
98      <bid_date>1999-01-25</bid_date>
99      </bid_tuple>
100
101   <bid_tuple>
102     <userid>U05</userid>
103     <itemno>1007</itemno>
104     <bid>200</bid>
105     <bid_date>1999-02-08</bid_date>
106     </bid_tuple>
107
108   <bid_tuple>
109     <userid>U04</userid>
110     <itemno>1007</itemno>
111     <bid>225</bid>
112     <bid_date>1999-02-12</bid_date>
113     </bid_tuple>
114 </bids>
```

Datei “items.xml”:

```
1     <items>
2       <item_tuple>
3         <itemno>1001</itemno>
4         <description>Red Bicycle</description>
5         <offered_by>U01</offered_by>
6         <start_date>19990105</start_date>
7         <end_date>19990120</end_date>
8         <reserve_price>40</reserve_price>
9       </item_tuple>
10      <item_tuple>
11        <itemno>1002</itemno>
12        <description>Motorcycle</description>
13        <offered_by>U02</offered_by>
14        <start_date>19990211</start_date>
15        <end_date>19990315</end_date>
16        <reserve_price>500</reserve_price>
17      </item_tuple>
18      <item_tuple>
19        <itemno>1003</itemno>
20        <description>Old Bicycle</description>
21        <offered_by>U02</offered_by>
22        <start_date>19990110</start_date>
23        <end_date>19990220</end_date>
```

```
      <reserve_price>25</reserve_price>
    </item_tuple>
    <item_tuple>
      <itemno>1004</itemno>
      <description>Tricycle</description>
    <offered_by>U01</offered_by>
      <start_date>19990225</start_date>
      <end_date>19990308</end_date>
      <reserve_price>15</reserve_price>
    </item_tuple>
    <item_tuple>
      <itemno>1005</itemno>
      <description>Tennis Racket</description>
      <offered_by>U03</offered_by>
      <start_date>19990319</start_date>
      <end_date>19990430</end_date>
      <reserve_price>20</reserve_price>
    </item_tuple>
    <item_tuple>
      <itemno>1006</itemno>
      <description>Helicopter</description>
    <offered_by>U03</offered_by>
      <start_date>19990505</start_date>
      <end_date>19990525</end_date>
      <reserve_price>50000</reserve_price>
    </item_tuple>
    <item_tuple>
      <itemno>1007</itemno>
      <description>Racing Bicycle </description>
      <offered_by>U04</offered_by>
      <start_date>19990120</start_date>
      <end_date>19990220</end_date>
      <reserve_price>200</reserve_price>
    </item_tuple>
    <item_tuple>
      <itemno>1008</itemno>
      <description>Broken Bicycle</description>
      <offered_by>U01</offered_by>
      <start_date>19990205</start_date>
      <end_date>19990306</end_date>
      <reserve_price>25</reserve_price>
    </item_tuple>
  </items>
```

5.1.4.2 Anfragen

Q1: *List the item number and description of all bicycles that currently have an auction in progress, ordered by item number.*

Lösung mit XQuery:

```
<result>
  {
    for $i in doc("items.xml")//item_tuple
    where $i/start_date <= current-date()
      and $i/end_date >= current-date()
      and contains($i/description, "Bicycle")
    order by $i/itemno
    return
        <item_tuple>
```

```
            { $i/itemno }
            { $i/description }
        </item_tuple>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all item_tupel {
    itemno {
      var Itemno
    },
    var Descr
  } ordering [Itemno] numeric
}
FROM
in {
  resource { "file:items.xml", "xml" },
  items {{
    item_tuple {{
      itemno {{ var Itemno }},
      var Descr -> description {{
        /.*Bicycle.*/
      }},
      start_date { var Startdate::date },
      end_date { var Enddate::date }
    }}
  }}
}
where {
  and {
    var Startdate::date lt current-date(),
    var Enddate::date gt current-date()
  }
}
END
```

Q2: *For all bicycles, list the item number, description, and highest bid (if any), ordered by item number.*

Lösung mit XQuery:

```
<result>
  {
    for $i in doc("items.xml")//item_tuple
    let $b := doc("bids.xml")//bid_tuple[itemno = $i/itemno]
    where contains($i/description, "Bicycle")
    order by $i/itemno
    return
        <item_tuple>
            { $i/itemno }
            { $i/description }
            <high_bid>{ max($b/bid) }</high_bid>
        </item_tuple>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all item_tupel {
    itemno {
      var Itemno
    },
    var Description,
    optional bid {
      max ( all var Bid )
    }
  } ordering [Itemno] numeric
}
FROM
and {
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        itemno {{ var Itemno }},
        var Description -> description {{ /.*Bicycle.*/ }}
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      optional bid_tuple {{
        itemno {{ var Itemno }},
         bid { var Bid }
      }}
    }}
  }
}
END
```

Q3: *Find cases where a user with a rating worse (alphabetically, greater) than "C" is offering an item with a reserve price of more than 1000.*

Lösung mit XQuery:

```
<result>
  {
    for $u in doc("users.xml")//user_tuple
    for $i in doc("items.xml")//item_tuple
    where $u/rating > "C"
       and $i/reserve_price > 1000
       and $i/offered_by = $u/userid
    return
        <warning>
            { $u/name }
            { $u/rating }
            { $i/description }
            { $i/reserve_price }
        </warning>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all warning {
    user { var Name },
    rating { var Rating::string },
    reserve_price { var Rp::float }
  }
}
FROM
and {
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid { var Userid },
        name { var Name },
        rating { var Rating::string }
      }}
    }} where { var Rating::string gt "C" }
  },
  items {{
    item_tuple {{
      reserve_price { var Rp::float },
      offered_by { var Userid }
    }}
  }} where { var Rp::float gt 1000 }
}
END
```

Q4: *List item numbers and descriptions of items that have no bids.*

Lösung mit XQuery:

```
<result>
  {
    for $i in doc("items.xml")//item_tuple
    where empty(doc("bids.xml")//bid_tuple[itemno = $i/itemno])
    return
        <no_bid_item>
            { $i/itemno }
            { $i/description }
        </no_bid_item>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all no_bid_item {
    itemno { var Itemno },
    description { var Description }
  }
}
FROM
```

```
and {
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        itemno { var Itemno },
        description { var Description }
      }}
    }}
  },
  not in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        itemno { var Itemno }
      }}
    }}
  }
}
END
```

Q5: *For bicycle(s) offered by Tom Jones that have received a bid, list the item number, description, highest bid, and name of the highest bidder, ordered by item number.*

Lösung mit XQuery:

```
<result>
  {
    for $seller in doc("users.xml")//user_tuple,
        $buyer in  doc("users.xml")//user_tuple,
        $item in  doc("items.xml")//item_tuple,
        $highbid in  doc("bids.xml")//bid_tuple
    where $seller/name = "Tom Jones"
      and $seller/userid  = $item/offered_by
      and contains($item/description , "Bicycle")
      and $item/itemno  = $highbid/itemno
      and $highbid/userid  = $buyer/userid
      and $highbid/bid = max(
                              doc("bids.xml")//bid_tuple
                                [itemno = $item/itemno]/bid
                          )
    order by ($item/itemno)
    return
        <jones_bike>
            { $item/itemno }
            { $item/description }
            <high_bid>{ $highbid/bid }</high_bid>
            <high_bidder>{ $buyer/name }</high_bidder>
        </jones_bike>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all jones_bike {
    var Itemno,
```

```
    var Description,
    bid {
      high_bidder { var Highbidder },
      high_bid { var Highbid }
    }
  }
}
FROM
and {
  temp1 {{
    jones_bike {{
      var Itemno -> itemno {{}},
      var Description -> description {{}},
      bidandbidder {{
        high_bidder { var Highbidder },
        bid { var Highbid }
      }}
    }}
  }} ,
  temp2 {{
    maxbid { var Highbid }
  }}
}
END

CONSTRUCT
temp2 {
  maxbid {
    max ( all var Maxbid )
  }
}
FROM
temp1 {{
  jones_bike {{
    bidandbidder {{
      bid { var Maxbid }
    }}
  }}
}}
END

CONSTRUCT
temp1 {
  all jones_bike {
    itemno { var Itemno },
    var Description,
    all bidandbidder {
      high_bidder { var Buyername },
      bid {
        max ( all var Bid )
      }
    }
  }
}
FROM
and {
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        name { "Tom Jones" },
        userid { var Userid }
      }}
```

```
    }}
  },
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        offered_by { var Userid },
        var Description -> description {/.*Bicycle.*/},
        itemno { var Itemno }
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        itemno { var Itemno },
        userid { var Buyerid },
        bid { var Bid }
      }}
    }}
  },
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid { var Buyerid },
        name { var Buyername }
      }}
    }}
  }
}
END
```

Q6: *For each item whose highest bid is more than twice its reserve price, list the item number, description, reserve price, and highest bid.*

Lösung mit XQuery:

```
<result>
  {
    for $item in doc("items.xml")//item_tuple
    let $b := doc("bids.xml")//bid_tuple[itemno = $item/itemno]
    let $z := max($b/bid)
    where $item/reserve_price * 2 < $z
    return
        <successful_item>
            { $item/itemno }
            { $item/description }
            { $item/reserve_price }
            <high_bid>{$z }</high_bid>
         </successful_item>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
high_bids {
  all high_bid_tuple {
    var Itemno,
    high_bid {
      max ( all var Bid )
    }
  }
}
FROM
in {
  resource { "file:bids.xml", "xml" },
  bids {{
    bid_tuple {{
      var Itemno -> itemno {{}},
      bid { var Bid }
    }}
  }}
}
END

CONSTRUCT
result {
  all successful_item {
    itemno { var Itemno },
    description { var Description },
    reserve_price { var Res::float },
    high_bid { var Highbid::float }
  }
}
FROM
and {
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        itemno { var Itemno },
        description { var Description },
        reserve_price { var Res::float }
      }}
    }}
  },
  high_bids {{
    high_bid_tuple {{
      itemno { var Itemno },
      high_bid { var Highbid::float }
      }
    }}
  }}
} where {var Res::float gt mult(var Highbid::float, 2) }
END
```

Q7: *Find the highest bid ever made for a bicycle or tricycle.*

Lösung mit XQuery:

```
let $allbikes := doc("items.xml")//item_tuple
                    [contains(description, "Bicycle")
                     or contains(description, "Tricycle")]
let $bikebids := doc("bids.xml")//bid_tuple[itemno = $allbikes/itemno]
return
    <high_bid>
      {
        max($bikebids/bid)
      }
    </high_bid>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all high_bid {
    high_bid {
      max ( all var Highbid )
    }
  }
}
FROM
and {
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        itemno { var Itemno },
        and [
          var Des -> description {/.*Bicycle.*/},
          var Des -> description {/.*Tricycle.*/}
        ]
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        itemno {{ var Itemno }},
        bid { var Highbid }
      }}
    }}
  }
}
END
```

Q8: *How many items were actioned (auction ended) in March 1999?*

Lösung mit XQuery:

```
let $item := doc("items.xml")//item_tuple
  [end_date >= xs:date("1999-03-01") and end_date <= xs:date("1999-03-31")]
return
    <item_count>
      {
        count($item)
      }
    </item_count>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  item_count {
    count ( all var Enddate::date )
  }
}
FROM
in {
  resource [ "file:items.xml", "xml" ],
  items {{
    item_tuple {{
      end_date { var Enddate::date }
    }}
  }}
}
where {
  and {
    var Enddate::date geq 1999-03-01,
    var Enddate::date leq 1999-03-31
  }
}
END
```

Q9: *List the number of items auctioned each month in 1999 for which data is available, ordered by month.*

Lösung mit XQuery:

```
<result>
  {
    let $end_dates := doc("items.xml")//item_tuple/end_date
    for $m in distinct-values(for $e in $end_dates
                              return get-month-from-date($e))
    let $item := doc("items.xml")
        //item_tuple[get-year-from-date(end_date) = 1999
                     and get-month-from-date(end_date) = $m]
    order by $m
    return
        <monthly_result>
            <month>{ $m }</month>
            <item_count>{ count($item) }</item_count>
        </monthly_result>
```

```
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
  result {
    all monthly_result {
      month { var Month },
      item_count {
        count ( all var Item )
      }
    } ordering [Month] literal
  }
FROM
  in {
    resource [ "file:items.xml", "xml" ],
    desc var Item -> item_tuple {{
      end_date { /1999-(var Month::integer -> [01][0-9]-([0-3][0-9])/}
    }}
  }
```

Q10: *For each item that has received a bid, list the item number, the highest bid, and the name of the highest bidder, ordered by item number.*

Lösung mit XQuery:

```
<result>
 {
    for $highbid in doc("bids.xml")//bid_tuple,
        $user in doc("users.xml")//user_tuple
    where $user/userid = $highbid/userid
      and $highbid/bid = max(doc("bids.xml")//bid_tuple[itemno=$highbid/itemno]/bid)
    order by $highbid/itemno
    return
        <high_bid>
            { $highbid/itemno }
            { $highbid/bid }
            <bidder>{ $user/name/text() }</bidder>
        </high_bid>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
high_bids {
  all high_bid_tuple {
    itemno { var Itemno },
    high_bid {
      max ( all toFloat( var Bid ))
    }
  }
}
FROM
in {
```

```
12    resource { "file:bids.xml", "xml" },
13    bids {{
14      bid_tuple {{
15        itemno { var Itemno },
16        bid { var Bid }
17      }}
18    }}
19  }
20  END
21
22  CONSTRUCT
23  bids {
24    all bid_tuple_with_username {
25      itemno { var Itemno },
26      bid { toFloat( var Bid ) },
27      name { var Name }
28    }
29  }
30  FROM
31  and {
32    in {
33      resource { "file:bids.xml", "xml" },
34      bids {{
35        bid_tuple {{
36          itemno { var Itemno },
37          bid { var Bid },
38          userid { var Userid }
39        }}
40      }}
41    },
42    in {
43      resource { "file:users.xml", "xml" },
44      users {{
45        user_tuple {{
46          userid { var Userid },
47          name { var Name }
48        }}
49      }}
50    }
51  }
52  END
53
54  CONSTRUCT
55  result {
56    all high_bid {
57      itemno { var Itemno },
58      bid { var Bid::float },
59      bidder { var Name }
60    } ordering [Itemno] literal
61  }
62  FROM
63  and {
64    bids {{
65      bid_tuple_with_username {{
66        itemno { var Itemno },
67        bid { var Bid::float },
68        name { var Name }
69      }}
70    }},
71    high_bids {{
72      high_bid_tuple {{
73        itemno { var Itemno },
74        high_bid { var Bid::float }
```

```
    }}
  }}
}
END
```

Q11: *List the item number and description of the item(s) that received the highest bid ever recorded, and the amount of that bid.*

Lösung mit XQuery:

```
let $highbid := max(doc("bids.xml")//bid_tuple/bid)
return
    <result>
     {
        for $item in doc("items.xml")//item_tuple,
            $b in doc("bids.xml")//bid_tuple[itemno = $item/itemno]
        where $b/bid = $highbid
        return
            <expensive_item>
                { $item/itemno }
                { $item/description }
                <high_bid>{ $highbid }</high_bid>
            </expensive_item>
     }
    </result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
highest_bids {
  max ( all var Bid )
}
FROM
in {
  resource { "file:bids.xml", "xml" },
  bids {{
    bid_tuple {{
      bid { var Bid }
    }}
  }}
}
END

CONSTRUCT
items {
  all items_with_description {
    itemno { var Itemno },
    description { var Description },
    high_bid {
      max ( all var Bid )
    }
  }
}
FROM
and {
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
```

```
      bid_tuple {{
        itemno { var Itemno },
        bid { var Bid }
      }}
    }}
  },
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        itemno { var Itemno },
        description { var Description }
      }}
    }}
  }
}
END

CONSTRUCT
result {
  all expensive_item {
    itemno { var Itemno },
    bid { var Bid },
    description { var Description }
  }
}
FROM
and {
  highest_bids { var Bid },
  items {{
    items_with_description {{
      itemno { var Itemno },
      description { var Description },
      high_bid { var Bid }
    }}
  }}
}
END
```

Q12: *List the item number and description of the item(s) that received the largest number of bids, and the number of bids it (or they) received.*

Lösung mit XQuery:

```
declare function local:bid_summary()
  as element()*
{
    for $i in distinct-values(doc("bids.xml")//itemno)
    let $b := doc("bids.xml")//bid_tuple[itemno = $i]
    return
        <bid_count>
            <itemno>{ $i }</itemno>
            <nbids>{ count($b) }</nbids>
        </bid_count>
};

<result>
 {
    let $bid_counts := local:bid_summary(),
        $maxbids := max($bid_counts/nbids),
```

```
17          $maxitemnos := $bid_counts[nbids = $maxbids]
18      for $item in doc("items.xml")//item_tuple,
19          $bc in $bid_counts
20      where $bc/nbids =  $maxbids and $item/itemno = $bc/itemno
21      return
22          <popular_item>
23              { $item/itemno }
24              { $item/description }
25              <bid_count>{ $bc/nbids/text() }</bid_count>
26          </popular_item>
27  }
28  </result>
```

Lösung mit Xcerpt:

```
1   CONSTRUCT
2   temp1 {
3     all bid_count {
4       itemno { var Itemno },
5       count {
6         toFloat( count ( all var Bid ) )
7       }
8     }
9   }
10  FROM
11  in {
12    resource { "file:bids.xml", "xml" },
13    bids {{
14      bid_tuple {{
15        itemno { var Itemno },
16        bid { var Bid }
17      }}
18    }}
19  }
20  END
21
22  CONSTRUCT
23  temp2 {
24    max_bid {
25      max ( all var Maxbid )
26    }
27  }
28  FROM
29  temp1 {{
30    bid_count {{
31      count { var Maxbid }
32    }}
33  }}
34  END
35
36  CONSTRUCT
37  result {
38    all items_with_max_bids {
39      itemno { var Itemno },
40      description { var Description },
41      count { var Maxbid }
42    }
43  }
44  FROM
45  and {
46    temp2 {{
```

```
    max_bid { var Maxbid }
  }} ,
  temp1 {{
    bid_count {{
      itemno { var Itemno },
      count { var Maxbid }
    }}
  }},
  in {
    resource { "file:items.xml", "xml" },
    items {{
      item_tuple {{
        itemno { var Itemno },
        description { var Description }
      }}
    }}
  }
}
END
```

Q13: *For each user who has placed a bid, give the userid, name, number of bids, and average bid, in order by userid.*

Lösung mit XQuery:

```
<result>
 {
    for $uid in distinct-values(doc("bids.xml")//userid),
        $u in doc("users.xml")//user_tuple[userid = $uid]
    let $b := doc("bids.xml")//bid_tuple[userid = $uid]
    order by $u/userid
    return
        <bidder>
            { $u/userid }
            { $u/name }
            <bidcount>{ count($b) }</bidcount>
            <avgbid>{ avg($b/bid) }</avgbid>
        </bidder>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
temp1 {
  all bidder {
    userid { var Userid },
    bidcount {
      count ( all var Bid )
    },
    avgbid {
      avg ( all var Bid )
    }
  }
}
FROM
in {
  resource { "file:bids.xml", "xml" },
```

```
  bids {{
    bid_tuple {{
      userid { var Userid },
      bid { var Bid }
    }}
  }}
}
END

CONSTRUCT
result {
  all bidder {
    userid { var Userid },
    name { var Name },
    bidcount { var Bidcount },
    avgbid { var Avgbid }
  } ordering [Userid] literal
}
FROM
and {
  temp1 {{
    bidder {{
      userid { var Userid },
      bidcount { var Bidcount },
      avgbid { var Avgbid }
    }}
  }},
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid { var Userid },
        name { var Name }
      }}
    }}
  }
}
END
```

Q14: *List item numbers and average bids for items that have received three or more bids, in descending order by average bid.*

Lösung mit XQuery:

```
<result>
 {
    for $i in distinct-values(doc("bids.xml")//itemno)
    let $b := doc("bids.xml")//bid_tuple[itemno = $i]
    let $avgbid := avg($b/bid)
    where count($b) >= 3
    order by $avgbid descending
    return
        <popular_item>
            <itemno>{ $i }</itemno>
            <avgbid>{ $avgbid }</avgbid>
        </popular_item>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
temp1 {
  all item {
    itemno { var Itemno  },
    bidcount {
      count ( all var Bid )
    },
    avgbid {
      avg ( all var Bid )
    }
  }
}
FROM
in {
  resource { "file:bids.xml", "xml" },
  bids {{
    bid_tuple {{
      itemno { var Itemno },
      bid { var Bid }
    }}
  }}
}
END

CONSTRUCT
result {
  all popular_item {
    itemno { var Itemno },
    avgbid { var Avgbid }
  } ordering [Avgbid] numeric
}
FROM
temp1 {{
  item {{
    itemno { var Itemno },
    bidcount { var Bidcount::integer },
    avgbid { var Avgbid }
  }}
}}
where { var Bidcount::integer gt 2 }
END
```

Q15: *List names of users who have placed multiple bids of at least $100 each.*

Lösung mit XQuery:

```
<result>
  {
    for $u in doc("users.xml")//user_tuple
    let $b := doc("bids.xml")//bid_tuple[userid=$u/userid and bid>=100]
    where count($b) > 1
    return
        <big_spender>{ $u/name/text() }</big_spender>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
temp1 {
  all bidder {
    userid { var Userid },
    bidcount {
      count ( all var Bid::integer )
    }
  }
}
FROM
bids {{
  bid_tuple {{
    userid { var Userid },
    bid { var Bid }
  }}
}}
where { var Bid::integer gt 100 }
END

CONSTRUCT
result {
  all big_spender { var Name }
}
FROM
and {
  temp1 {{
    bidder {{
      userid { var Userid },
      bidcount { var Bidcount::integer }
    }}
  }}
  where { var Bidcount::integer gt 1 } ,
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        name { var Name },
        userid { var Userid }
      }}
    }}
  }
}
END
```

Q16: *List all registered users in order by userid; for each user, include the userid, name, and an indication of whether the user is active (has at least one bid on record) or inactive (has no bid on record).*

Lösung mit XQuery:

```
<result>
  {
    for $u in doc("users.xml")//user_tuple
    let $b := doc("bids.xml")//bid_tuple[userid = $u/userid]
    order by $u/userid
    return
        <user>
```

```
            { $u/userid }
            { $u/name }
            {
                if (empty($b))
                  then <status>inactive</status>
                  else <status>active</status>
            }
        </user>
  }
</result>
```

1. Lösung mit Xcerpt:

```
CONSTRUCT
   var State
FROM
   var State -> state {
       status {"active"},
       status {"inactive"}
   }
END

CONSTRUCT
result {
  all user {
    userid {var Userid},
    name {var Name}
  }
}
FROM
and {
  in {
    resource { "file:../users.xml", "xml" },
    users {{
      user_tuple {{
        userid {var Userid},
        name {var Name}
      }}
    }}
  },
  in {
    resource { "file:../bids.xml", "xml" },
    bids {{
      bid_tuple {{
        IF
          userid {var Userid}
        THEN
          state {{
            var Status -> status {"active"}
          }}
        ELSE
          state {{
            var Status -> status {"inactive"}
          }}
      }}
    }}
  }
}
```

2. Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all user {
    userid {var Userid},
    name {var Name},
    var Status
  }
}
FROM
and {
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid {var Userid},
        name {var Name}
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        IF
          userid {var Userid}
        THEN
          var Status :< status {"active"}
        ELSE
          var Status :< status {"inactive"}
       }}
    }}
  }
}
```

Q17: *List the names of users, if any, who have bid on every item.*

Lösung mit XQuery:

```
<frequent_bidder>
  {
    for $u in doc("users.xml")//user_tuple
    where
      every $item in doc("items.xml")//item_tuple satisfies
        some $b in doc("bids.xml")//bid_tuple satisfies
          ($item/itemno = $b/itemno and $u/userid = $b/userid)
    return
        $u/name
  }
</frequent_bidder>
```

Lösung mit Xcerpt:

```
CONSTRUCT
temp1 {
  all user {
    userid { var Userid },
    name { var Name },
    all itemno { var Itemno },
    itemcount {
      count ( all var Itemno )
    }
  }
}
FROM
and {
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid { var Userid },
        name { var Name }
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
        userid { var Userid },
        itemno { var Itemno }
      }}
    }}
  }
}
END

CONSTRUCT
temp2 {
  itemcount {
    count ( all var Item )
  }
}
FROM
in {
  resource { "file:../items.xml", "xml" },
  items {{
    var Item -> item_tuple {{}}
  }}
}
END

CONSTRUCT
result {
  all frequent_bidder {
    optional var Name with default "no_bidder"
  }
}
FROM
and {
  temp1 {{
    user {{
      name { optional var Name },
```

```
      itemcount { var Itemcount }
    }}
  }},
  temp2 {{
    itemcount { var Itemcount }
  }}
}
END
```

Q18: *List all users in alphabetic order by name. For each user, include descriptions of all the items (if any) that were bid on by that user, in alphabetic order.*

Lösung mit XQuery:

```
<result>
  {
    for $u in doc("users.xml")//user_tuple
    order by $u/name
    return
        <user>
            { $u/name }
            {
                for $b in distinct-values(doc("bids.xml")//bid_tuple
                                          [userid = $u/userid]/itemno)
                for $i in doc("items.xml")//item_tuple[itemno = $b]
                let $descr := $i/description/text()
                order by $descr
                return
                    <bid_on_item>{ $descr }</bid_on_item>
            }
        </user>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
temp1 {
  all user {
    name { var Name },
    all itemno { var Itemno }
  } ordering [Name] literal
}
FROM
and {
  in {
    resource { "file:users.xml", "xml" },
    users {{
      user_tuple {{
        userid { var Userid },
        name { var Name }
      }}
    }}
  },
  in {
    resource { "file:bids.xml", "xml" },
    bids {{
      bid_tuple {{
```

```
23          userid { var Userid },
24          itemno { var Itemno }
25        }}
26      }}
27    }
28  }
29  END
30
31  CONSTRUCT
32  result {
33    all user {
34      name { var Name },
35      all bid_on_item {
36        var Description
37      } ordering [Description] literal
38    }
39  }
40  FROM
41  and {
42    in {
43      resource { "file:items.xml", "xml" },
44      items {{
45        item_tuple {{
46          itemno { var Itemno },
47          description { var Description }
48        }}
49      }}
50      },
51    temp1 {{
52      user {{
53        name { var Name },
54        itemno { var Itemno }
55      }}
56    }}
57  }
58  END
```

5.1.5 Use Case “SGML: Standard Generalized Markup Language”

5.1.5.1 Quelldatei

Datei “report.xml”:

```
<!DOCTYPE report SYSTEM "report.dtd">
<report>
  <title>Getting started with SGML</title>
  <chapter>
    <title>The business challenge</title>
    <intro>
      <para>With the ever-changing and growing global market, companies and
        large organizations are searching for ways to become more viable and
        competitive. Downsizing and other cost-cutting measures demand more
        efficient use of corporate resources. One very important resource is
        an organization's information.</para>
      <para>As part of the move toward integrated information management,
        whole industries are developing and implementing standards for
        exchanging technical information. This report describes how one such
        standard, the Standard Generalized Markup Language (SGML), works as
        part of an overall information management strategy.</para>
      <graphic graphname="infoflow"/></intro></chapter>
  <chapter>
    <title>Getting to know SGML</title>
    <intro>
      <para>While SGML is a fairly recent technology, the use of
        <emph>markup</emph> in computer-generated documents has existed for a
        while.</para></intro>
    <section shorttitle="What is markup?">
      <title>What is markup, or everything you always wanted to know about
        document preparation but were afraid to ask?</title>
      <intro>
        <para>Markup is everything in a document that is not content. The
          traditional meaning of markup is the manual <emph>marking</emph> up
          of typewritten text to give instructions for a typesetter or
          compositor about how to fit the text on a page and what typefaces to
          use. This kind of markup is known as <emph>procedural markup</emph>.
         </para>
      </intro>
      <topic topicid="top1">
        <title>Procedural markup</title>
        <para>Most electronic publishing systems today use some form of
          procedural markup. Procedural markup codes are good for one
          presentation of the information.</para></topic>
      <topic topicid="top2">
        <title>Generic markup</title>
        <para>Generic markup (also known as descriptive markup) describes the
          <emph>purpose</emph> of the text in a document. A basic concept of
          generic markup is that the content of a document must be separate from
          the style. Generic markup allows for multiple presentations of the
          information.</para></topic>
      <topic topicid="top3">
        <title>Drawbacks of procedural markup</title>
        <para>Industries involved in technical documentation increasingly
          prefer generic over procedural markup schemes. When a company changes
          software or hardware systems, enormous data translation tasks arise,
          often resulting in errors.</para></topic></section>
    <section shorttitle="What is SGML?">
      <title>What <emph>is</emph> SGML in the grand scheme of the universe, anyway?
      </title>
```

```
56        <intro>
57          <para>SGML defines a strict markup scheme with a syntax for defining
58            document data elements and an overall framework for marking up
59            documents.</para>
60          <para>SGML can describe and create documents that are not dependent on
61            any hardware, software, formatter, or operating system. Since SGML
62        documents conform to an international standard, they are portable.
63        </para>
64        </intro>
65      </section>
66      <section shorttitle="How does SGML work?">
67        <title>How is SGML and would you recommend it to your grandmother?</title>
68        <intro>
69          <para>You can break a typical document into three layers: structure,
70            content, and style. SGML works by separating these three aspects and
71            deals mainly with the relationship between structure and content.</para>
72        </intro>
73        <topic topicid="top4">
74          <title>Structure</title>
75          <para>At the heart of an SGML application is a file called the DTD, or
76            Document Type Definition. The DTD sets up the structure of a document,
77            much like a database schema describes the types of information it
78            handles.</para>
79          <para>A database schema also defines the relationships between the
80            various types of data. Similarly, a DTD specifies <emph>rules</emph>
81            to help ensure documents have a consistent, logical structure.</para>
82        </topic>
83        <topic topicid="top5">
84          <title>Content</title>
85          <para>Content is the information itself. The method for identifying
86            the information and its meaning within this framework is called
87            <emph>tagging</emph>. Tagging must
88            conform to the rules established in the DTD (see <xref xrefid="top4"/>).
89      </para>
90          <graphic graphname="tagexamp"/></topic>
91        <topic topicid="top6">
92          <title>Style</title>
93          <para>SGML does not standardize style or other processing methods for
94            information stored in SGML.</para></topic></section></chapter>
95    <chapter>
96      <title>Resources</title>
97      <section>
98        <title>Conferences, tutorials, and training</title>
99        <intro>
100         <para>The Graphic Communications Association has been
101           instrumental in the development of SGML. GCA provides conferences,
102           tutorials, newsletters, and publication sales for both members and
103           non-members.</para>
104         <para security="c">Exiled members of the former Soviet Union's secret
105           police, the KGB, have infiltrated the upper ranks of the GCA and are
106           planning the Final Revolution as soon as DSSSL is completed.</para>
107       </intro>
108     </section>
109   </chapter>
110 </report>
```

5.1.5.2 Anfragen

Q1: *Locate all paragraphs in the report (all "para" elements occurring anywhere within the "report" element).*

Lösung mit XQuery:

```
<result>
  {
    doc("sgml.xml")//report//para
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all var Para
]
FROM
in {
  resource { "file:report.xml", "xml" },
  desc var Para -> para {{}}
}
END
```

Q2: *Locate all paragraph elements in an introduction (all "para" elements directly contained within an "intro" element).*

Lösung mit XQuery:

```
<result>
  {
    doc("sgml.xml")//intro/para
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all var Para
]
FROM
in {
  resource { "file:report.xml", "xml" },
  desc intro [[
    var Para -> para {{}}
  ]]
}
END
```

Q3: *Locate all paragraphs in the introduction of a section that is in a chapter that has no introduction (all ”para” elements directly contained within an ”intro” element directly contained in a ”section” element directly contained in a ”chapter” element. The ”chapter” element must not directly contain an ”intro” element).*

Lösung mit XQuery:

```
<result>
  {
    for $c in doc("sgml.xml")//chapter
    where empty($c/intro)
    return $c/section/intro/para
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all var Para
]
FROM
in {
  resource { "file:report.xml", "xml" },
  desc chapter {{
    without intro {{}},
    section [[
      intro [[
        var Para -> para {{}}
      ]]
    ]]
  }}
}
END
```

Q4: *Locate the second paragraph in the third section in the second chapter (the second “para” element occurring in the third “section” element occurring in the second “chapter” element occurring in the “report”).*

Lösung mit XQuery:

```
<result>
  {
    (((doc("sgml.xml")//chapter)[2]//section)[3]//para)[2]
  }
</result>
```

1. Lösung mit Xcerpt:

```
CONSTRUCT
  temp1 [
    first (drop 1 (all var Chapter ))
  ]
```

```
FROM
  in {
    resource { "file:report.xml", "xml" },
    report [[
      var Chapter -> chapter [[]]
    ]]
  }
END

CONSTRUCT
  temp2 [
    first ( drop 2 (all var Section ))
  ]
FROM
  temp1 [[
    desc var Section -> section {{}}
  ]]
END

CONSTRUCT
  result [
    first (drop 1 ( all var Para ))
  ]
FROM
  temp2 [[
    desc var Para -> para {{}}
  ]]
END
```

2. Lösung mit Xcerpt:

```
CONSTRUCT
temp [
  all meta_chapter [
    var Chapter,
    all meta_section [
      var Section,
      all meta_para [
        var Para
      ]
    ]
  ]
]
FROM
in {
  resource { "file:report.xml", "xml" },
  report [[
    var Chapter -> chapter [[
      var Section -> section [[
        var Para -> para {{}}
      ]]
    ]]
  ]]
}
END

CONSTRUCT
  result [
    var Para
  ]
FROM
```

```
  temp [[
  position 2 meta_chap [[
    position 4 meta_section [[
      position 3 meta_para {{ var Para }}
  }}
  ]]
END

CONSTRUCT
  result [
    first (drop 1 ( all var Para ))
  ]
FROM
  temp2 [[
    desc var Para -> para {{}}
  ]]
END
```

Q5: *Locate all classified paragraphs (all ”para” elements whose ”security” attribute has the value ”c”).*

Lösung mit XQuery:

```
<result>
  {
    doc("sgml.xml")//para[@security = "c"]
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all var Para
]
FROM
in {
  resource { "file:report.xml", "xml" },
  desc var Para -> para [[
    attributes  {{
      security [ "c" ]
    }}
  ]]
}
END
```

Q6: *List the short titles of all sections (the values of the ”shorttitle” attributes of all ”section” elements, expressing each short title as the value of a new element.)*

Lösung mit XQuery:

```
<result>
  {
    for $s in doc("sgml.xml")//section/@shorttitle
    return <stitle>{ $s }</stitle>
```

```
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
  results [
    all stitle [ var STitle ]
  ]
FROM
  in {
    resource { "file:report.xml", "xml" },
    desc section [[
      attributes  {{
        shorttitle [ var STitle ]
      }}
    ]]
  }
END
```

Q7: *Locate the initial letter of the initial paragraph of all introductions (the first character in the content [character content as well as element content] of the first "para" element contained in an "intro" element).*

Lösung mit XQuery:

```
<result>
  {
    for $i in doc("sgml.xml")//intro/para[1]
    return
        <first_letter>{ substring(string($i), 1, 1) }</first_letter>
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
temp [
  all var Intro [
    first ( all var Para )
  ]
]
FROM
in {
  resource { "file:report.xml", "xml" },
  desc /(var Intro -> intro)/ [[
    var Para -> para [[]]
  ]]
}
END

CONSTRUCT
result [
  all var Para {
    first_letter {
```

```
      first (all var FirstLetter )
    }
  }
]
FROM
temp [[
  desc /(var Para -> para)/ [[
    desc /(var FirstLetter -> .).*/
  ]]
]]
END
```

Q8a: *Locate all sections with a title that has “is SGML” in it. The string may occur anywhere in the descendants of the title element, and markup boundaries are ignored.*

Lösung mit XQuery:

```
<result>
   {
     doc("sgml.xml")//section[.//title[contains(., "is SGML")]]
   }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
  temp [
    all section_title {
      var Section,
      content {
        toText (var Content)
      }
    }
  ]
FROM
  in {
    resource { "file:report.xml", "xml" },
    desc var Section -> section [[
      title {{ var Content }
    ]]
  }
END

CONSTRUCT
  result [ all var Section ]
FROM
  section_title [[
    var Section -> section {{}},
    content {{ /.*is SGML.*/
  ]]
END
```

Q8b: *Same as (Q8a), but the string ”is SGML” cannot be interrupted by sub-elements, and must appear in a single text node.*

Lösung mit XQuery:

```
<result>
   {
     doc("sgml.xml")//section[.//title/text()[contains(., "is SGML")]]
   }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  all var Section
]
FROM
in {
  resource { "file:report.xml", "xml" },
  desc var Section -> section [[
    title {{ /.*is SGML.*/ }}
  ]]
}
END
```

Q9: *Locate all the topics referenced by a cross-reference anywhere in the report (all the "topic" elements whose "topicid" attribute value is the same as an "xrefid" attribute value of any "xref" element).*

Lösung mit XQuery:

```
<result>
  {
    for $id in doc("sgml.xml")//xref/@xrefid
    return doc("sgml.xml")//topic[@topicid = $id]
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
  result [ all var Topic ]
FROM
  in {
    resource { "file:report.xml", "xml" },
    and {
      desc var Topic -> topic [[
        attributes  {{
          topicid [ var XRefID ]
        }}
      ]],
      desc xref [[
        attributes  {{
          xrefid [ var XRefID ]
        }}
      ]]
    }
  }
END
```

Q10: *Locate the closest title preceding the cross-reference (”xref”) element whose ”xrefid” attribute is ”top4” (the ”title” element that would be touched last before this ”xref” element when touching each element in document order).*

Lösung mit XQuery:

```
<result>
  {
    let $x := doc("sgml.xml")//xref[@xrefid = "top4"],
        $t := doc("sgml.xml")//title[. << $x]
    return $t[last()]
  }
</result>
```

Lösung mit Xcerpt:

```
CONSTRUCT
results [
  last ( all var Title )
]
FROM
in {
  resource { "file:report.xml", "xml" },
  desc var X [[
    desc title {{
      var Title
    }},
    desc xref {
      attributes  {
        xrefid {"top4"}
      }
    }
  ]]
}
END
```

5.1.6 Use Case “STRING: String Search”

5.1.6.1 Quelldateien

Datei “company_data.xml”:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE company SYSTEM "company.dtd">
<company>
   <name>Foobar Corporation</name>
   <ticker_symbol>FOO</ticker_symbol>

   <description>Foobar Corporation is a maker of Foo(TM) and
        Foobar(TM) products and a leading software company with a 300
        Billion dollar revenue in 1999. It is located in Alaska.
   </description>

   <business_code>Software</business_code>
   <partners>
        <partner>YouNameItWeIntegrateIt.com</partner>
        <partner>TheAppCompany Inc.</partner>
   </partners>
   <competitors>
        <competitor>Gorilla Corporation</competitor>
   </competitors>
</company>
```

Datei “input.xml”:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<news>
<news_item>
   <title> Gorilla Corporation acquires YouNameItWeIntegrateIt.com </title>
   <content>
      <par> Today, Gorilla Corporation announced that it will purchase
          YouNameItWeIntegrateIt.com. The shares of
          YouNameItWeIntegrateIt.com dropped $3.00 as a result of this
          announcement.
      </par>
      <par> As a result of this acquisition, the CEO of
          YouNameItWeIntegrateIt.com Bill Smarts resigned. He did not
          announce what he will do next.  Sources close to
          YouNameItWeIntegrateIt.com hint that Bill Smarts might be
          taking a position in Foobar Corporation.
      </par>
      <par>YouNameItWeIntegrateIt.com is a leading systems integrator
          that enables <quote>brick and mortar</quote> companies to
          have a presence on the web.
      </par>
   </content>
   <date>1-20-2000</date>
   <author>Mark Davis</author>
   <news_agent>News Online</news_agent>
</news_item>

<news_item>
   <title>Foobar Corporation releases its new line of Foo products
   today</title>
```

```
   <content>
      <par> Foobar Corporation releases the 20.9 version of its Foo
            products.  The new version of Foo products solve known
            performance problems which existed in 20.8 line and
            increases the speed of Foo based products tenfold. It also
            allows wireless clients to be connected to the Foobar
            servers.
      </par>
      <par> The President of Foobar Corporation announced that they
            were proud to release 20.9 version of Foo products and
            they will upgrade existing customers <footnote>where
            service agreements exist</footnote>
            promptly. TheAppCompany Inc. immediately announced that it
            will release the new version of its products to utilize
            the 20.9 architecture within the next three months.
      </par>
      <figure>
          <title>Presidents of Foobar Corporation and TheAppCompany
          Inc. Shake Hands</title> <image source="handshake.jpg"/>
      </figure>
   </content>
   <date>1-20-2000</date>
   <news_agent>Foobar Corporation</news_agent>
</news_item>

<news_item> <title>Foobar Corporation is suing Gorilla Corporation for
   patent infringement </title>
   <content>
      <par> In surprising developments today, Foobar Corporation
         announced that it is suing Gorilla Corporation for patent
         infringement. The patents that were mentioned as part of the
         lawsuit are considered to be the basis of Foobar
         Corporation's <quote>Wireless Foo</quote> line of products.
      </par>
      <par>The tension between Foobar and Gorilla Corporations has
         been increasing ever since the Gorilla Corporation acquired
         more than 40 engineers who have left Foobar Corporation,
         TheAppCompany Inc. and YouNameItWeIntegrateIt.com over the
         past 3 months. The engineers who have left the Foobar
         corporation and its partners were rumored to be working on
         the next generation of server products and applications which
         will directly compete with Foobar's Foo 20.9 servers. Most of
         the engineers have relocated to Hawaii where the Gorilla
         Corporation's server development is located.
      </par>
   </content>
   <date>1-20-2000</date>
   <news_agent>Reliable News Corporation</news_agent>
</news_item>
</news>
```

5.1.6.2 Anfragen

Q1: *Find the titles of all news items where the string "Foobar Corporation" appears in the title.*

Lösung mit XQuery:

```
doc("string.xml")//news_item/title[contains(., "Foobar Corporation")]
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all var Title
}
FROM
in {
  resource { "file:input.xml", "xml" },
  desc news_item {{
    var Title -> title {/.*Foobar Corporation.*/}
  }}
}
END
```

Q2: *Find news items where the Foobar Corporation and one or more of its partners are mentioned in the same paragraph and/or title. List each news item by its title and date.*

Lösung mit XQuery:

```
declare function local:partners($company as xs:string) as element()*
{
    let $c := doc("company-data.xml")//company[name = $company]
    return $c//partner
};

let $foobar_partners := local:partners("Foobar Corporation")

for $item in doc("string.xml")//news_item
where
  some $t in $item//title satisfies
    (contains($t/text(), "Foobar Corporation")
    and (some $partner in $foobar_partners satisfies
      contains($t/text(), $partner/text())))
  or (some $par in $item//par satisfies
   (contains(string($par), "Foobar Corporation")
     and (some $partner in $foobar_partners satisfies
        contains(string($par), $partner/text()))))
return
    <news_item>
        { $item/title }
        { $item/date }
    </news_item>
```

Lösung mit Xcerpt:

```
CONSTRUCT
temp {
  all news_item {
    var News
  }
}
FROM
and {
  in {
    resource { "file:company_data.xml", "xml" },
    company {{
```

```
      name {"Foobar Corporation"},
      desc partner {var Partner}
    }}
  },
  in {
    resource { "file:input.xml", "xml" },
    desc news_item {{
      or {
        and {
          title {/.*Foobar Corporation.*/},
          title {/.*(var Partner ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/}
        },
        and {
          desc par {/.*Foobar Corporation.*/},
          desc par {/.*(var Partner ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/}
        }
      }}
    }
  }
}
END

CONSTRUCT
result {
  all news_item {
    var Title,
    var Date
  }
}
FROM
temp {{
  news_item {{
    var Title -> title{{}},
    var Date -> date {{}}
  }}
}}
END
```

Q3: *Query Q3 has been withdrawn from the use cases document.*

Q4: *Find news items where a company and one of its partners is mentioned in the same news item and the news item is not authored by the company itself.*

Lösung mit XQuery:

```
declare function local:partners($company as xs:string) as element()*
{
    let $c := doc("company-data.xml")//company[name = $company]
    return $c//partner
};

for $item in doc("string.xml")//news_item,
    $c in doc("company-data.xml")//company
let $partners := local:partners($c/name)
where contains(string($item), $c/name)
  and (some $p in $partners satisfies
    contains(string($item), $p) and $item/news_agent != $c/name)
return
    $item
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all var News_Item
}
FROM
and {
  in {
    resource { "file:company_data.xml", "xml" },
    company {{
      name {var Company},
      desc partner {var Partner}
    }}
  },
  in {
    resource { "file:input.xml", "xml" },
    and {
      desc var News_Item -> news_item {{
        and {
          desc /.*(var Company ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/,
          desc /.*(var Partner ->[A-Z][a-zA-Z]*( [A-Z][a-zA-Z])?).*/
        },
        without news_agent{ var Company }
      }}
    }
  }
}
END
```

Q5: *For each news item that is relevant to the Gorilla Corporation, create an ”item summary” element. The content of the item summary is the content of the title, date, and first paragraph of the news item, separated by periods. A news item is relevant if the name of the company is mentioned anywhere within the content of the news item.*

Lösung mit XQuery:

```
for $item in doc("string.xml")//news_item
where contains(string($item/content), "Gorilla Corporation")
return
    <item_summary>
        { $item/title/text() }
        { $item/date/text() }
        { string(($item//par)[1]) }
    </item_summary>
```

Lösung mit Xcerpt:

```
CONSTRUCT
result {
  all item_summary {
    title { var Title },
    date { var Date },
    first ( all var Par )
  }
}
```

```
FROM
in {
  resource { "file:input.xml", "xml" },
  desc news_item {{
    content {{
      and [
        desc var G -> /.*Gorilla Corporation.*/,
        desc var Par -> par {{}}
      ]
    }},
    title { var Title },
    date { var Date }
  }}
}
END
```

5.1.7 Use Case “NS: Queries Using Namespaces”

5.1.7.1 Quelldatei

Datei “reviews.xml”:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<ma:AuctionWatchList
    xmlns:ma="http://www.example.com/AuctionWatch"
    xmlns:xlink="http://www.w3.org/1999/xlink"
    xmlns:anyzone="http://www.example.com/auctioneers#anyzone"
    xmlns:eachbay="http://www.example.com/auctioneers#eachbay"
    xmlns:yabadoo="http://www.example.com/auctioneers#yabadoo"
>

<!-- ____________________________________________________________________ -->

<ma:Auction anyzone:ID="0321K372910">

    <ma:AuctionHomepage
        xlink:type="simple"
        xlink:href="http://www.example.com/item/0321K372910"
    />

    <ma:Schedule>
        <ma:Open    xmlns:dt="http://www.w3.org/2001/XMLSchema"
                dt:type="timeInstant">2000-03-21:07:41:34-05:00</ma:Open>
        <ma:Close    xmlns:dt="http://www.w3.org/2001/XMLSchema"
                dt:type="timeInstant">2000-03-23:07:41:34-05:00</ma:Close>
    </ma:Schedule>

    <ma:Price>
        <ma:Start ma:currency="USD">3.00</ma:Start>
        <ma:Current ma:currency="USD">10.00</ma:Current>
        <ma:Number_of_Bids>5</ma:Number_of_Bids>
    </ma:Price>

    <ma:Trading_Partners>
        <ma:High_Bidder>
            <eachbay:ID>RecordsRUs</eachbay:ID>
            <eachbay:PositiveComments>231</eachbay:PositiveComments>
            <eachbay:NeutralComments>2</eachbay:NeutralComments>
            <eachbay:NegativeComments>5</eachbay:NegativeComments>
            <ma:MemberInfoPage
                xlink:type="simple"
                xlink:href="http://auction.eachbay.com/members?get=RecordsRUs"
                xlink:role="ma:MemberInfoPage"
            />
        </ma:High_Bidder>
        <ma:Seller>
            <anyzone:ID>VintageRecordFreak</anyzone:ID>
            <anyzone:Member_Since>October 1999</anyzone:Member_Since>
            <anyzone:Rating>5</anyzone:Rating>
            <ma:MemberInfoPage
               xlink:type="simple"
               xlink:href="http://auction.anyzone.com/members/VintageRecordFreak"
               xlink:role="ma:MemberInfoPage"
            />
        </ma:Seller>
    </ma:Trading_Partners>
```

```
56
57      <ma:Details>
58          <record xmlns="http://www.example.org/music/records">
59              <artist>Miles Davis</artist>
60              <title>In a Silent Way</title>
61              <recorded>1969</recorded>
62              <label>Columbia Records</label>
63
64              <remark>
65                  With Miles Davis (trumpet), Herbie Hancock (Electric
66                  Piano), Chick Corea (Electric Piano), Wayne Shorter
67                  (Tenor Sax), Josef Zawinul (Electric Piano &amp;amp;
68                  Organ), John McLaughlin (Guitar), and Tony Williams
69                  (Drums).  The liner notes were written by Frank Glenn,
70                  and the record is in fine condition.
71              </remark>
72
73          </record>
74      </ma:Details>
75
76  </ma:Auction>
77
78  <!-- ____________________________________________________________________ -->
79
80
81  <ma:Auction yabadoo:ID="13143816">
82
83      <ma:AuctionHomepage
84          xlink:type="simple"
85          xlink:href="http://auctions.yabadoo.com/auction/13143816"
86      />
87
88      <ma:Schedule>
89          <ma:Open    xmlns:dt="http://www.w3.org/2001/XMLSchema"
90                  dt:type="timeInstant">2000-03-19:17:03:00-04:00</ma:Open>
91          <ma:Close    xmlns:dt="http://www.w3.org/2001/XMLSchema"
92                  dt:type="timeInstant">2000-03-29:17:03:00-04:00</ma:Close>
93      </ma:Schedule>
94
95      <ma:Price>
96          <ma:Start ma:currency="USD">3.00</ma:Start>
97          <ma:Current ma:currency="USD">3.00</ma:Current>
98          <ma:Number_of_Bids>0</ma:Number_of_Bids>
99      </ma:Price>
100
101     <ma:Trading_Partners>
102         <ma:High_Bidder>
103             <eachbay:ID>VintageRecordFreak</eachbay:ID>
104             <eachbay:PositiveComments>232</eachbay:PositiveComments>
105             <eachbay:NeutralComments>0</eachbay:NeutralComments>
106             <eachbay:NegativeComments>0</eachbay:NegativeComments>
107             <ma:MemberInfoPage
108                 xlink:type="simple"
109                 xlink:href="http://auction.eachbay.com/showRating/user=VintageRecordFreak"
110                 xlink:role="ma:MemberInfoPage"
111             />
112         </ma:High_Bidder>
113         <ma:Seller xmlns:seller="http://www.example.com/auctioneers#eachbay">
114             <seller:ID>StarsOn45</seller:ID>
115             <seller:PositiveComments>80</seller:PositiveComments>
116             <seller:NeutralComments>1</seller:NeutralComments>
117             <seller:NegativeComments>2</seller:NegativeComments>
118             <ma:MemberInfoPage
```

```
                xlink:type="simple"
                xlink:href="http://auction.eachbay.com/showRating/user=StarsOn45"
                xlink:role="ma:MemberInfoPage"
            />
        </ma:Seller>
    </ma:Trading_Partners>

    <ma:Details>
        <record xmlns="http://www.example.org/music/records">
            <artist>Wynton Marsalis</artist>
            <title>Think of One ...</title>
            <recorded>1983</recorded>
            <label>Columbia Records</label>
            <remark xml:lang="en"> Columbia Records 12" 33-1/3 rpm LP,
                #FC-38641, Stereo. The record is still clean and shiny
                and looks unplayed (looks like NM condition).  The
                cover has very light surface and edge wear.
            </remark>
            <remark xml:lang="de"> Columbia Records 12" 33-1/3 rpm LP,
                #FC-38641, Stereo. Die Platte ist noch immer sauber
                und glänzend und sieht ungespielt aus
                (NM Zustand). Das Cover hat leichte Abnutzungen an
                Oberfläche und Ecken.
            </remark>
        </record>
    </ma:Details>

</ma:Auction>

</ma:AuctionWatchList>
```

5.1.7.2 Anfragen

Q1: *List all unique namespaces used in the sample data.*

Lösung mit XQuery:

```
<Q1>
  {
    for $n in distinct-values(
                  for $i in (doc("auction.xml")//* | doc("auction.xml")//@*)
                  return namespace-uri($i)
                )
    return  <ns>{$n}</ns>
  }
</Q1>
```

Lösung mit Xcerpt:

```
CONSTRUCT
Q1 {
  all ns {
    var NS
  }
}
FROM
in {
  resource {"file:ns.xml","xml"},
```

```
  desc var NS:/.*/ {{}}
}
END
```

Q2: *Select the title of each record that is for sale.*

Lösung mit XQuery:

```
declare namespace music = "http://www.example.org/music/records";

<Q2>
  {
    doc("auction.xml")//music:title
  }
</Q2>
```

Lösung mit Xcerpt:

```
ns-prefix music="http://www.example.org/music/records"

CONSTRUCT
Q2 {
  all var T
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc var T -> music:title {{}}
}
END
```

Q3: *Select all elements that have an attribute whose name is in the XML Schema namespace.*

Lösung mit XQuery:

```
declare namespace dt = "http://www.w3.org/2001/XMLSchema";

<Q3>
  {
    doc("auction.xml")//*[@dt:*]
  }
</Q3>
```

Lösung mit Xcerpt:

```
ns-prefix dt="http://www.w3.org/2001/XMLSchema"

CONSTRUCT
Q3 {
  all var Element
}
FROM
in {
```

```
  resource {"file:ns.xml","xml"},
  desc var Element -> /.*/:/.*/ {{
    attributes {{
      dt:/.*/ {{}}
    }}
  }}
}
END
```

Q4: *List the target URI's of all XLinks in the document.*

Lösung mit XQuery:

```
declare namespace xlink = "http://www.w3.org/1999/xlink";

<Q4 xmlns:xlink="http://www.w3.org/1999/xlink">
  {
    for $hr in doc("auction.xml")//@xlink:href
    return <ns>{ $hr }</ns>
  }
</Q4>
```

Lösung mit Xcerpt:

```
ns-prefix xlink="http://www.w3.org/1999/xlink"

CONSTRUCT
Q4 {
  all ns {
    var Href
  }
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc /.*/ {{
    attributes {{
      var Href -> xlink:href {{}}
    }}
  }}
}
END
```

Q5: *Select all records that have a remark in German.*

Lösung mit XQuery:

```
declare namespace music = "http://www.example.org/music/records";

<Q5 xmlns:music="http://www.example.org/music/records">
  {
     doc("auction.xml")//music:record[music:remark/@xml:lang = "de"]
  }
</Q5>
```

Lösung mit Xcerpt:

```
ns-prefix music="http://www.example.com/music/records"

CONSTRUCT
Q5 {
  all var R
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc var R -> music:record {{
    music:remark {{
      attributes {{
        xml:lang{"de"}
      }}
    }}
  }}
}
END
```

Q6: *Select the closing time elements of all AnyZone auctions currently monitored.*

Lösung mit XQuery:

```
declare namespace ma = "http://www.example.com/AuctionWatch";
declare namespace anyzone = "http://www.example.com/auctioneers#anyzone";

<Q6 xmlns:ma="http://www.example.com/AuctionWatch">
  {
    doc("auction.xml")//ma:Auction[@anyzone:ID]/ma:Schedule/ma:Close
  }
</Q6>
```

Lösung mit Xcerpt:

```
ns-prefix ma="http://www.example.com/AuctionWatch"
ns-prefix anyzone="http://www.example.com/auctioneers#anyzone"

CONSTRUCT
Q6 {
  all var Close
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc var R -> ma:Auctions {{
    attributes {{
      anyzone:ID {{}}
    }},
    desc var Close -> ma:Close {{}}
  }}
}
END
```

Q7: *Select the homepage of all auctions where both seller and high bidder are registered at the same auctioneer.*

Lösung mit XQuery:

```
declare namespace ma = "http://www.example.com/AuctionWatch";

<Q7 xmlns:xlink="http://www.w3.org/1999/xlink">
  {
    for $a in doc("auction.xml")//ma:Auction
    let $seller_id := $a/ma:Trading_Partners/ma:Seller/*:ID,
        $buyer_id := $a/ma:Trading_Partners/ma:High_Bidder/*:ID
    where namespace-uri($seller_id) = namespace-uri($buyer_id)
    return
        $a/ma:AuctionHomepage
  }
</Q7>
```

Lösung mit Xcerpt:

```
ns-prefix ma="http://www.example.com/AuctionWatch"

CONSTRUCT
Q7 {
  all var Homepage
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc  ma:Auctions {{
    var Homepage -> ma:AuctionHomepage {{}},
    and {
      desc ma:HighBidder {{
        var NS:ID {{}}
      }},
      desc ma:Seller {{
        var NS:ID {{}}
      }}
    }
  }}
}
END
```

Q8: *Select all traders (either seller or high bidder) without negative comments.*

Lösung mit XQuery:

```
declare namespace ma = "http://www.example.com/AuctionWatch";

<Q8 xmlns:ma="http://www.example.com/AuctionWatch"
    xmlns:eachbay="http://www.example.com/auctioneers#eachbay"
    xmlns:xlink="http://www.w3.org/1999/xlink">
  {
    for $s in doc("auction.xml")//ma:Trading_Partners/(ma:Seller | ma:High_Bidder)
    where $s/*:NegativeComments = 0
    return $s
  }
</Q8>
```

Lösung mit Xcerpt:

```
ns-prefix ma="http://www.example.com/AuctionWatch"

CONSTRUCT
Q7 {
  all var Homepage
}
FROM
in {
  resource {"file:ns.xml","xml"},
  desc  ma:Auctions {{
    var Homepage -> ma:AuctionHomepage {{}},
    and {
      ma:HighBidder {{
        var NS:ID {{}}
      }},
      ma:Seller {{
        var NS:ID {{}}
      }}
    }
  }}
}
END

```

5.1.8 Use Case “PARTS: Recursive Parts Explosion”

5.1.8.1 Quelldatei

Datei “partlist.xml”:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<partlist>
  <part partid="0" name="car"/>
  <part partid="1" partof="0" name="engine"/>
  <part partid="2" partof="0" name="door"/>
  <part partid="3" partof="1" name="piston"/>
  <part partid="4" partof="2" name="window"/>
  <part partid="5" partof="2" name="lock"/>
  <part partid="10" name="skateboard"/>
  <part partid="11" partof="10" name="board"/>
  <part partid="12" partof="10" name="wheel"/>
  <part partid="20" name="canoe"/>
</partlist>
```

5.1.8.2 Anfragen

Q1: *Convert the sample document from “partlist” format to “parttree” format (see DTD section for definitions). In the result document, part containment is represented by containment of one <part> element inside another. Each part that is not part of any other part should appear as a separate top-level element in the output document.*

Lösung mit XQuery:

```
declare function local:one_level($p as element()) as element()
{
    <part partid="{ $p/@partid }"
          name="{ $p/@name }" >
        {
            for $s in doc("partlist.xml")//part
            where $s/@partof = $p/@partid
            return local:one_level($s)
        }
    </part>
};

<parttree>
  {
    for $p in doc("partlist.xml")//part[empty(@partof)]
    return local:one_level($p)
  }
</parttree>
```

1. Lösung mit Xcerpt:

```
GOAL
var Result
FROM
and {
```

```
  in {
    resource { "file:partlist.xml", "xml" },
    var Root -> partlist {{}}
  },
  init [ var Root, var Result ]
}
END

CONSTRUCT
init [
  var In,
  result [
    all part {
      attributes {
        partid { var PartId },
        name { var Name }
      },
      optional all var ChildPartRes
    }
  ]
]
FROM
and {
  partlist {{
    part {
      attributes {
        partid { var PartId },
        name { var Name }
      }
    },
    optional var ChildPart -> part {
      attributes {
        partid {{}},
        partof { var PartId }
      }
    }
  }} :< var In,
  optional hasChild [ var ChildPart, var ChildPartRes ]
}
END

CONSTRUCT
hasChild [
  var In,
  part {
    attributes {
        partid { var Partid },
        name { var Name }
    },
    optional all var ChildPartRet
  }
]
FROM
and {
  part {{
    attributes {
      partid { var Partid },
      partof { var Partof },
      name { var Name }
    }
  }} :< var In,
    in {
    resource { "file:partlist.xml", "xml" },
```

```
68      optional partlist {{
69        var Child -> part {{
70          attributes {
71            partid {{}},
72            partof { var Partid  }
73          }
74        }}
75      }}
76    },
77    optional hasChild [ var Child, var ChildPartRet ]
78  }
79  END
```

2. Lösung mit Xcerpt:

```
1   CONSTRUCT
2   parttree {
3     all var PartId1@part {
4       attributes {
5         name { var Name }
6       },
7       optional all ^var PartId2
8     }
9   }
10  FROM
11  in {
12    resource { "file:partlist.xml", "xml" },
13    partlist {{
14      part {
15        attributes {
16          partid { var PartId1 },
17          optional partof { var Partof },
18          name { var Name }
19        }
20      },
21      optional part {
22        attributes {
23          partid { var PartId2 },
24          partof { var PartId1 },
25          name { var Name2 }
26        }
27      }
28    }}
29  }
30  END
```

5.2 Mediatoren Beispieldokumente

5.2.1 Dokument: “dblp.xml”

```
<?xml version="1.0" encoding="iso-8859-1"?>
<article key="BryHM03">
<author>François Bry</author>
<author>Nicola Henze</author>
<author>Jan Maluszynski</author>
<title>Principles and Practice of Semantic Web Reasoning</title>
<year>2003</year>
</article>

<article key="BergerBS03">
<author>Sacha Berger</author>
<author>François Bry</author>
<author>Sebastian Schaffert</author>
<title>A Visual Language for Web Querying and Reasoning</title>
<pages>99-112</pages>
<year>2003</year>
</article>

<article key="BergerBSW03">
<author> Sacha Berger</author>
<author>François Bry</author>
<author>Sebastian Schaffert</author>
<author>Christoph Wieser</author>
<title>Xcerpt and visXcerpt: From Pattern-Based to Visual
       Querying of XML and Semistructured Data
</title>
<pages>1053-1056</pages>
<year>2003</year>
</article>

<article key="OlteanuMFB02">
<author>Dan Olteanu</author>
<author>Holger Meuss</author>
<author>Tim Furche</author>
<author>François Bry</author>
<title>XPath: Looking Forward</title>
<pages> 109-127</pages>
<year>2002</year>
</article>

<article key="SchaffertB02">
<author>Sebastian Schaffert</author>
<author>François Bry</author>
<title>A gentle introduction to Xcerpt, a rule-based query and
       transformation language for XML
</title>
<year>2002</year>
</article>

<article key="Ohlbach99">
<author>Hans Jürgen Ohlbach</author>
<title>Set Description Languages and Reasoning about Numerical
       Features of Sets
</title>
<year>1999</year>
</article>

```

```
<article key="GottlobSR96">
<author>Georg Gottlob</author>
<author>Michael Schrefl</author>
<author>Brigitte Röck</author>
<title>Extending Object-Oriented Systems with Roles.</title>
<pages>268-296</pages>
<year>1996</year>
</article>
```

5.2.2 Dokument: "amazon.xml"

```
<?xml version="1.0" encoding="iso-8859-1"?>
<productinfo>
        <details url="http://www.amazon.de/exec/obidos/ASIN/3540637923">
                <asin>3540637923</asin>
                <productname>Deductive and Object-Oriented Databases</productname>
                <catalog>Book</catalog>
                <authors>
                        <author>François Bry</author>
                        <author>Raghu Ramakrishnan</author>
                        <author>Kotagiri Ramamohanarao</author>
                </authors>
                <releasedate>December 1997</releasedate>
                <manufacturer>Springer</manufacturer>
                <ourprice>EUR 55,64</ourprice>
        </Details>
</productinfo>
<productinfo>
        <details url="http://www.amazon.de/exec/obidos/ASIN/3540205829">
                <asin>3540205829</asin>
                <productname>Principles and Practice of Semantic Web Reasoning
                </productname>
                <catalog>Book</catalog>
                <authors>
                        <author>François Bry</author>
                        <author>Nicola Henze</author>
                        <author>Jan Maluszynski</author>
                </authors>
                <releasedate>Dezember 2003</releasedate>
                <manufacturer>Springer</manufacturer>
                <ourprice>EUR 40,66</ourprice>
        </details>
</productinfo>
<productinfo>
        <details url="http://www.amazon.de/exec/obidos/ASIN/3884579711">
                <asin>3884579711</asin>
                <productname>Workshop (14.) Logische Programmierung</productname>
                <catalog>Book</catalog>
                <authors>
                        <author>François Bry </author>
                        <author>Ulrich Geske</author>
                        <author>Dietmar Seipel</author>
                </authors>
                <releasedate>Januar 2000</releasedate>
                <manufacturer></manufacturer>
                <ourprice></ourprice>
        </details>
</productinfo>
```

5.2.3 Dokument: “pms.html”

```
<?xml version="1.0" encoding="iso-8859-1"?>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
          "http://www.w3.org/TR/2000/REC-xhtml1-20000126/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <body>
    <p>
      <i>Lehr- und Forschungseinheit für
         Programmier- und Modellierungssprachen,
         <br />
         Institut für Informatik der Ludwig-Maximilians-Universität München
      </i>
    </p>
    <hr />

    <h2><a name="ResearchPapers">Research Papers</a></h2>
    <h3>2004</h3>
    <ul>
      <li>
        <a name="PMS-FB-2004-1">PMS-FB-2004-1</a>
        <br />
        Dan Olteanu, Tim Furche, François Bry:
        <br />

        <em>An Efficient Single-Pass Query Evaluator for XML Data Streams</em>.
        <br />
        <a href="PMS-FB/PMS-FB-2004-1.abstract">Abstract (.txt)</a>,
        <a href="PMS-FB/PMS-FB-2004-1.ps.gz">complete text (.ps.gz, 673K)</a>,
        <a href="PMS-FB/PMS-FB-2004-1.pdf"> (.pdf, 191K)</a>.
        <br />
        In: Proceedings of <a href="http://www.acm.org/conferences/sac/sac2004/">
        19th Annual ACM Symposium on Applied Computing (SAC)</a>
        (<a href="http://www.lsi.us.es/%7Eaguilar/ds/">Data Streams Track</a>),
        Nicosia/Cyprus, 14-17 March 2004
        <p></p>
      </li>
      <li>
        <a name="PMS-FB-2004-2">PMS-FB-2004-2</a>
        <br />
        François Bry, Tim Furche, Dan Olteanu:
        <br />
        <em>Datenströme</em>.
        <br />

        <a href="PMS-FB/PMS-FB-2004-2.abstract">Abstract (.txt)</a>,
        <a href="PMS-FB/PMS-FB-2004-2.ps.gz">complete text (.ps.gz, 465K)</a>,
        <a href="PMS-FB/PMS-FB-2004-2.pdf"> (.pdf, 101K)</a>.
        <br />
        In: Informatik Spektrum, Volume 27, Number 2, April 2004,
        <br />
        &copy; <a href="http://www.springer.de/comp/lncs/">Springer-Verlag</a>
        <p></p>

      </li>
    </ul>
    <h3>2003</h3>
    <ul>
     <li>
        <a name="PMS-DISS-2003-1">PMS-DISS-2003-1</a>
        <br />
```

```
61          Michael Marte:
62          <br />
63          <em>Models and Algorithms for School Timetabling -
64              A Constraint-Programming Approach (Dissertation)
65          </em>.
66          <br />
67          <a href="PMS-FB/PMS-DISS-2003-1.abstract">Abstract (.txt)</a>,
68          <a href="PMS-FB/PMS-DISS-2003-1.ps.gz">complete text (.ps.gz, 265K)</a>,
69          <a href="PMS-FB/PMS-DISS-2003-1.pdf">(.pdf, 602K</a>).
70          <br />
71          <p></p>
72        </li>
73        <li>
74          <a name="PMS-FB-2003-1">PMS-FB-2003-1</a>
75          <br />
76          Sacha Berger, François Bry, Sebastian Schaffert:
77          <br />
78          <em>Pattern Queries for XML and Semistructured Data</em>.
79          <br />
80
81          <a href="PMS-FB/PMS-FB-2003-1.abstract">Abstract (.txt)</a>,
82          <a href="PMS-FB/PMS-FB-2003-1.ps.gz">complete text (.ps.gz, 541K)</a>,
83          <a href="PMS-FB/PMS-FB-2003-1.pdf"> (.pdf, 318K)</a>,
84          <a href="PMS-FB/PMS-FB-2002-5/WLP02-BrySchaffert-Slides.pdf">Slides (.pdf)</a>.
85          <br /><!--
86          In:
87          <a href="http://www.computational-logic.org/local/wlp2002/cfp_wlp_2002.html">
88             17. Workshop Logische Programmierung</a>,
89          Dresden, 11.-13. December 2002-->
90          In: <a href="http://www.computational-logic.org/local/wlp2002/TUD-FI03-03.ps.gz">
91                    Proceedings</a> of the
92              <a href="http://www.computational-logic.org/local/wlp2002/cfp_wlp_2002.html">
93                    17. Workshop Logische Programmierung</a>,
94                    Dresden, 11.-13. December 2002
95          <br />
96          Revision of Research Report <a href="#PMS-FB-2002-5">PMS-FB-2002-5</a>
97          <p></p>
98        </li>
99        <li>
100         <a name="PMS-FB-2003-6">PMS-FB-2003-6</a>
101         <br />
102         Sacha Berger, François Bry, Sebastian Schaffert:
103         <br />
104         <em>A Visual Language for Web Querying and Reasoning</em>.
105         <br />
106             <a href="PMS-FB/PMS-FB-2003-6.abstract">Abstract (.txt)</a>,
107         <a href="PMS-FB/PMS-FB-2003-6.pdf">complete text (.pdf, 320K)</a>
108         <br />
109         In: <a href="http://www.kbs.uni-hannover.de/~henze/ppswr03/">
110             Workshop on Principles and Practice of Semantic Web Reasoning</a> at
111             ICLP03, Mumbai, India, December 2003
112         <br />
113         <p></p>
114       </li>
115       <li>
116         <a name="PMS-FB-2003-2">PMS-FB-2003-2</a>
117         <br />
118         Sacha Berger, François Bry, Sebastian Schaffert, Christoph Wieser:
119         <br />
120         <em>Xcerpt and visXcerpt: From Pattern-Based to Visual Querying of XML and
121             Semistructured Data
122         </em>.
123         <br />
```

```
124         <a href="PMS-FB/PMS-FB-2003-2.pdf">complete text (.pdf, 180K)</a>
125         <br />
126         In: Proceedings of the <a href="http://www.vldb.informatik.hu-berlin.de/">
127         29th Intl. Conference on Very Large Databases (VLDB03)</a>,
128             Berlin/Germany, September 2003
129         <br />
130         <p></p>
131       </li>
132       </ul>
133 </body>
134 </html>
135
```

Literaturverzeichnis

[ADMFDF+98] Alin Deutsch, Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu. *XML-QL: A Query Language for XML.* W3 Consortium, http://www.w3.org/TR/1998/NOTE-xml-ql-19980819, 1998. 8

[BDS95] P. Buneman, S. Davidson, and D. Suciu. Programming constructs for unstructured data. In *DBLP*, 1995. 6

[DCDDMF+03] Don Chamberlin, Denise Draper, Mary Fernández, Micheal Kay, Jonathan Robie, Michael Rys, Jérôme Siméon, Jim Tivy, and Philip Wadler. *XQuery from the Experts. A Guide to the W3C XML Query Language.* Addison-Wesley, 2003. 6

[DCF01] David C. Fallside. *XML Schema Part 0: Primer. W3C Recommendation.* W3 Consortium, http://www.w3.org/TR/xmlschema-0, 2001. 78

[DCPFDF+03] Don Chamberlin, Peter Fankhauser, Daniela Florescu, Massimo Marchiori, and Jonathan Robie. *XML Query Use Cases. W3C Working Draft.* W3 Consortium, http://www.w3.org/TR/xquery-use-cases, 2003. 40, 41

[DRALHIJ99] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. *HTML 4.01 Specification. W3C Recommendation.* W3 Consortium, http://www.w3.org/TR/html401, 1999. 24

[FBSS02a] François Bry and Sebastian Schaffert. A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In *International Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-11, 2002. 6, 7

[FBSS02b] François Bry and Sebastian Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In *Web, Web-Services, and Database Systems*, http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-7, 2002. 8

[FBSS02c] François Bry and Sebastian Schaffert. Towards a Declarative Query and Transformation Language for XML and Semistructured Data: Simulation Unification. In *International Conference on Logic Programming (ICLP)*, http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2002-2, 2002. 7

[JC99] James Clark. *XSL Transformations (XSLT) Version 1.0. W3C Recommendation.* W3 Consortium, http://www.w3.org/TR/xslt, 1999. 6

[JCSD99] James Clark and Steve DeRose. *XML Path Language (XPath).* W3 Consortium, http://www.w3.org/TR/xpath, 1999. 6

[LSAS99] Leon Sterling and Ahud Shapiro. *The Art of Prolog.* The MIT Press, 1999. 49

[SAPBDS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web. From Relations to Semistructured Data and XML.* Morgan Kaufmann Publishers, San Francisco, CA, 2000. 8

[SBDCMFF+03] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniela Florescu, Jonathan Robie, and Jérôme Siméon. *XQuery 1.0: An XML Query Language. W3C Working Draft.* W3 Consortium, http://www.w3.org/TR/xquery/, 2003. 6

[SBFBSS03] Sacha Berger, François Bry, and Sebastian Schaffert. A Visual Language for Web Querying and Reasoning. In *Workshop on Principles and Practice of Semantic Web Reasoning*, http://www.pms.informatik.uni-muenchen.de/publikationen/#PMS-FB-2003-6, 2003. 7

[xml00] W3 Consortium, http://www.w3.org/XML/. *XML Specification*, 2000. 24