BSA-Algebra für XQuery
Operatoren, Optimierungsregeln und Anwendungen

Alf-Christian Schering, Ammar S. Balouch, Andreas Heuer
Institut für Informatik, Universität Rostock
E-Mail: {acsg, ab006, ah}@informatik.uni-rostock.de

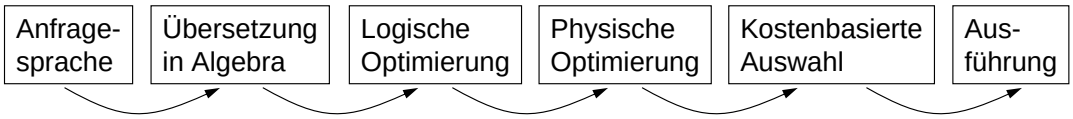
Zusammenfassung

Dieser Artikel stellt die BSA-Algebra vor, welche explizit für die formale Abbildung der Anfragesprache XQuery entwickelt wurde. Ähnlich wie bei relationalen Anfragesprachen fungiert eine Algebra auch bei XQuery als unverzichtbares Mittel zur formalen Anfragerepräsentation während der Verarbeitung und Optimierung. In Anbetracht der Tatsache, dass es bislang keinen Standard in punkto Algebra für XQuery gibt und die wenigen Ansätze ursprünglich meistens für andere semistrukturierte Anfragesprachen entwickelt wurden, ist die Entwicklung von BSA lohnenswert.

Die BSA-Algebra unterscheidet sich von anderen semistrukturierten XML-Algebren u. a. im verwendeten Datenmodell; wie XQuery so operiert auch BSA auf Sequenzen von Bäumen. Dazu wurden Operatoren definiert, welche sich zum Teil an der etablierten Relationenalgebra orientieren jedoch auch BSA-spezifische Operatoren beinhalten. Zu letzteren gehört beispielsweise der Typoperator, welcher Optimierungspotential im Zusammenhang mit statischer Typprüfung bietet.

1 Einleitung

Die algebraische XML-Anfrageverarbeitung und -optimierung orientiert sich an bekannten Konzepten und Vorgehensweisen aus dem Bereich der relationalen Datenbanken. Eines der Hauptziele sowohl der SQL- als auch der XQuery-Anfragebearbeitung ist es, eine effiziente Anfrageauswertung auf den gegebenen Daten durchzuführen. Wie eingangs erwähnt, bedient man sich dafür in beiden Fällen des Mittels der Optimierung. Diese kann im Allgemeinen auf zwei verschiedene Arten vorgenommen werden — *logisch* und *physisch*. Folgende Abbildung zeigt die generelle Vorgehensweise bei der Anfrageverarbeitung sowohl für relationale [4], als auch für XML-Daten:



Es ist zunächst eine mathematische Formalisierung der komplexen, textuellen Anfragesprache erforderlich. Wie bei SQL werden Anfragen erst einmal in algebraische Ausdrücke konvertiert und dann optimiert. An dieser Stelle kommt ein Vorschlag für eine semistrukturierte Algebra, nämlich BSA [3], zum Einsatz.

2 Die BSA-Algebra

Die Abkürzung *BSA*, eigentlich *βSA*, steht für ***B**eta **S**equenz **A**lgebra*.

Der griechische Buchstabe **Beta** bezeichnet den Operator der BSA-Algebra, der für die Bestimmung des Typs einer Eingabesequenz oder eines Ausgabeausdrucks verwendet wird. Die Typisierung ist ein Hauptbestandteil dieser BSA-Algebra. Unter anderem grenzt sich BSA, durch die Verwendung eines Typoperators und einer Typkomponente, wesentlich von anderen semistrukturierten XML-Algebren ab. Da XQuery-Ausdrücke praktisch immer typbehaftet sind, stellt diese Eigenschaft einen entscheidenden Vorteil dar.

Die BSA-Algebra verwendet Baumstrukturen zur Darstellung von XML-Dokumenten. Die Baumstrukturen selbst, können Elemente einer geordneten Menge (**Sequenz**) sein. Sequenzen werden von der BSA-Algebra als Trägermenge verwendet, auf denen die algebraischen Operatoren arbeiten (siehe Abschnitt 3).

Eine **Algebra** beschreibt mathematische Strukturen und die auf diesen definierten Verknüpfungen. Die Funktion der Strukturen übernehmen in BSA die Komponenten. Sie bilden die XQuery-Teilausdrücke ab, welche die XML-Fragmente beschreiben. Die Verknüpfungen entsprechen den BSA-Operatoren. Diese repräsentieren die „Operatoren“ der XQuery-Anfragesprache (also ihre Hauptausdrücke).

2.1 Eigenschaften von BSA

Die BSA-Algebra besitzt verschiedene Eigenschaften, die in direktem Zusammenhang mit denen der zugrunde liegenden Anfragesprache stehen. Eine optimale Abbildung ist nur dann möglich, wenn alle Eigenschaften der Anfragesprache auch in der Algebra berücksichtigt werden. [5] definiert in einer *generellen Forderung* für XML-Anfragealgebren wichtige Kriterien, wie Vollständigkeit, Abgeschlossenheit, Orthogonalität, Implementierbarkeit und Optimierbarkeit. [6] zeigt für jedes dieser Kriterien, dass es bezüglich der BSA-Algebra erfüllt ist.

3 Die BSA-Komponenten

Die Komponenten sind die Bestandteile der BSA-Algebra auf denen die Operatoren arbeiten[1]. Sie repräsentieren die Daten, über denen die zugrunde liegenden XQuery-Anfragen ausgeführt werden. Man unterscheidet sieben BSA-Komponenten: Eingabekomponente *Seq*, Ausgabekomponente *Expr*, Bedingungskomponente *Cond*, Typkomponente *Type*, Variablenkomponente *Var*, Namenskomponente *Name* und Sortierungskomponente *Sort*.

Der Eingabeteil eines XQuery-`FLWOR`-Ausdrucks verwendet `FOR`- und `LET`-Klauseln, um die Ausdrücke in deren `IN`-Klauseln an ihre jeweiligen Variablen zu binden. Im Eingabeteil des korrespondierenden BSA-Ausdrucks werden die `IN`-Klauseln durch Sequenzkomponenten (*Seq*) und die Bindungsvariablen durch Variablenkomponenten (*Var*) dargestellt. Variablenkomponenten können außerdem innerhalb von *Expr*-, *Seq*-, *Cond*- oder *Sort*-Komponenten auftreten, um dort die gebundenen Variablen zu referenzieren.

Die Ausgabekomponente spezifiziert die Ergebnisstruktur in primären BSA-Operatoren und bildet damit die `RETURN`-Klausel eines `FLWOR`-Ausdrucks ab. Die Namenskomponente wird verwendet, um Element- oder Attributknoten in der Ausgabe eines BSA-Operators zu benennen.

Bedingungskomponenten *Cond* werden in Operatoren verwendet, die in irgendeiner Form Selektionen durchführen. Sie repräsentieren dabei atomare Selektionsbedingungen. Es sind nur atomare Bedingungen, z.B. Vergleichs- oder quantifizierte Ausdrücke und deren aussagenlogische Kombinationen in Form einer konjunktiven Normalform zugelassen [6].

Aufgabe der Typkomponente *Type* ist es, den Typ von XQuery-Ausdrücken in den sie repräsentierenden BSA-Ausdrücken abzubilden. So werden z.B. die XQuery-Typen `element()` auf *ele*, `attribute()` auf *att* oder `xs:string` auf *str* abgebildet. Die Typkomponente hat meistens keine direkt sichtbare Entsprechung im zugrunde liegenden XQuery-Ausdruck, da Typinformationen dort i.d.R. implizit durch die Struktur der Anfragen vorhanden sind[2]. Folgendes Beispiel zeigt eine XQuery-Anfrage und die in ihr existierenden Komponenten.

[1] [6] definiert die vollständige grammatikalische Anbindung der BSA-Komponenten an die BSA-Operatoren.

[2] D.h. dass der Typ von XQuery-Fragmenten, die z.B. durch Pfadausdrücke oder Literale beschrieben sind, aus diesen direkt abgeleitet werden kann.

Beispiel XQuery-Anfrage:

```
FOR $s IN doc("Sterne.xml")//Stern
LET $d := 1234
WHERE $s/@distance > $d
RETURN
   <Stern spectral="Q">
     {$s/Name}
   </Stern>
```

BSA-Ausdruck:

$$\longrightarrow \quad s :: ele \otimes doc(''Sterne.xml'')//Stern$$
$$\longrightarrow \quad d :: int := 1234$$
$$\longrightarrow \quad \text{s/@distance} > \text{d}$$
$$\longrightarrow \quad Stern\langle spectral\langle ''Q'' :: str\rangle :: att, \; s/Name :: ele\rangle :: ele$$

4 Die BSA-Operatoren

Die *BSA-Operatoren*[3] werden in drei Gruppen unterteilt: Hauptoperatoren, interne und implizite Operatoren[4]. Die **Hauptoperatoren** stellen eine formale Repräsentation für die XQuery-Hauptausdrücke (`FLWOR`) dar. Zu ihnen gehören: Projektion Ψ, Rekonstruktion $\Re$, Selektion, Δ, Verbund χ und Typoperator β. Zu den internen Operatoren gehören die Sortierung ρ, die Duplikateliminierung λ und die Quantoroperatoren ε und $\wp$. Im Gegensatz zu den Hauptoperatoren können sie nicht orthogonal verwendet werden [1].

Projektionsoperator Ψ und Rekonstruktionsoperator $\Re$

Ähnlich wie in der Relationenalgebra, führt auch in BSA der Projektionsoperator auf bestimmte Teile der Elemente der Eingabesequenz eine Projektion durch. Das Ausgabeschema wird dabei bzgl. des Schemas der Eingabe eingeschränkt. Folgendes Beispiel zeigt eine Projektion:

```
FOR $a IN //A
RETURN <A>{$a/C}</A>
```

XQuery-Anfrage

$$\Psi^{A\langle a/C::ele\rangle::ele}(a :: ele \otimes //A)$$

BSA-Ausdruck

Der Rekonstruktionsoperator strukturiert die Ausgabe (bzgl. der Eingabe) neu. Durch die zusätzlichen Funktionalitäten Umbenennung und Strukturänderung unterscheidet er sich vom Projektionsoperator. In Tabelle 1 werden Syntax und Semantik der Hauptoperatoren beschrieben.

Selektionsoperator Δ

Der Selektionsoperator wählt bestimmte Elemente aus der Eingabesequenz aus. Zur Darstellung des Selektionskriteriums wird die in Abschnitt 3 definierte Bedingungskomponente *Cond* genutzt:

```
FOR $a IN //A
WHERE $a/@id>10
RETURN $a
```

XQuery-Anfrage

$$\Psi^{a::ele}(\mathbf{\Delta}_{a/@id>10}(a :: ele \otimes //A))$$

BSA-Ausdruck

Verbundoperator χ

Der Verbundoperator verbindet die Schemata mehrerer Eingabesequenzen miteinander. Er realisiert die Erzeugung des XQuery-Tupelstreams [7] und die Selektion der Tupel nach der Verbundbedingung. Die Verbundoperation ist vergleichbar mit dem natürlichen Verbund der Relationenalgebra. Folgendes Beispiel zeigt eine typische Join-Operation in XQuery und BSA:

```
FOR $a IN //A, $b IN //B
WHERE $a/@id = $b/@id
RETURN <AB>{$a, $b}</AB>
```

XQuery-Anfrage

$$\Re^{AB\langle a::ele,b::ele\rangle::ele}(\chi_{a/@id=b/@id}(a :: ele \otimes //A, b :: ele \otimes //B))$$

BSA-Ausdruck

[3] [1, 2, 6] beschreiben die Erzeugung der BSA-Operatoren aus XQuery-Ausdrücken detailliert.
[4] Alle XQuery-Operatoren sind implizit auch BSA-Operatoren.

Projektion:	$\Psi^{Name\langle Expr::Type\rangle::Type}($ $Var :: Type \otimes Seq)$	$\Psi^{Expr::type}(v :: type \otimes seq) := \{s(Expr) \mid s \preceq seq\}$
Rekonstruktion:	$\Re^{Name\langle Expr::Type\rangle::Type}($ $Var :: Type \otimes Seq)$	$\Re^{Expr::type}(v :: type \otimes seq) := \{s(Expr) \mid s \in seq\}$
Selektion:	$\Delta_{Cond}(Var :: Type \otimes Seq)$	$\Delta_C(v :: type \otimes seq) := \{s \mid s \in seq \wedge C(s)\}$
Verbund:	$\chi_{Cond}(Var_1 :: Type_1 \otimes Seq_1, \ldots Var_n :: Type_n \otimes Seq_n) \mid n \geq 2$	$\chi_{C_{s_1,s_2}}(v_1 :: type_1 \otimes seq_1, v_2 :: type_2 \otimes seq_2) := \{(s_1, s_2) \mid [s_1 \in seq_1 \wedge s_2 \in seq_2] \wedge C_{s_1,s_2}\}$
Kreuzprodukt (via χ):	$\chi_{true}\ (Var_1 :: Type_1 \otimes Seq_1, \ldots Var_n :: Type_n \otimes Seq_n) \mid n \geq 2$	$\chi_{true}(v_1 :: type_1 \otimes seq_1, v_2 :: type_2 \otimes seq_2) := \{(s_1, s_2) \mid s_1 \in seq_1 \wedge s_2 \in seq_2\}$
Typoperator:	Eingabe: $\beta(Seq)$,	$\beta(seq) := \{typeof(s) \mid s \in seq\}$,
	Ausgabe: $\beta(Expr)$,	$\beta(Expr) := \{typeof(e) \mid e \in Expr\}$,
	Bedingung: $\beta(Cond)$	$\beta(Cond) := \begin{cases} Boolean & \mid \beta(Op_1) <: \beta(Op_2) \\ \lightning & \mid sonst \end{cases}$

Tabelle 1: Syntax (links) und Semantik (rechts) der BSA-Hauptoperatoren

Typoperator β
Die allgegenwärtige Präsenz von Typinformationen ist integraler Bestandteil der BSA-Algebra. Typinformationen werden mit Hilfe des Typoperators und der Typkomponente dargestellt. Sie ermöglichen die Erschließung einer Menge weiterer Optimierungsregeln für BSA-Ausdrücke, welche sich Typinformationen zu Nutze machen. Ein Beispiel illustriert die Verwendung des Typoperators zur Bestimmung des Typs des `IF`-Ausdrucks in der `RETURN`-Klausel:

```
LET $i := 123, $s := "abc"
RETURN
  <E1>
    {IF(c) THEN $s ELSE $i}
  </E1>
```

XQuery-Anfrage

$$\Re^{E1\langle x_1::\beta(x_1)\rangle::ele}(\chi_{true}(i :: int := 123, s :: str := \text{''}abc\text{''}))\ x_1 = (s[c], i[not(c)])$$

BSA-Ausdruck

5 Optimierung

So wie in der relationalen Anfrageverarbeitung, bietet eine Algebra auch für XQuery die Möglichkeit einer formalen, logischen Anfrageoptimierung. Hierzu können auf Basis der BSA-Algebra Regeln definiert werden, die logische Redundanzen entfernen, verschachtelte Anfragen verflachen, heuristische Verbesserungen vornehmen oder triviale Ausdrücke auflösen. In [1, 2, 6] wurden bereits einige Anfrageoptimierungsregeln definiert. Folgender Abschnitt beschreibt eine Regel, welche bestimmte verschachtelte `FLWOR`-Ausdrücke erkennt und entfernt (hier der einfache Fall: direkte Verwendung von Eingabevariablen in der `RETURN`-Klausel des inneren `FLWOR`-Ausdrucks).

Verschachtelung innerhalb der Eingabe
Handelt es sich bei der äußeren Operation um eine Projektion oder eine Rekonstruktion, die nur eine Eingabesequenz mit dem inneren Projektionsoperator enthält, gilt folgende Regel:

$$\Psi^{out_o}(outer :: ele \otimes \Psi^{out_i}(inner :: ele \otimes seq)) \quad \Rightarrow \quad \Psi^{out}(inner :: ele \otimes seq)$$
$$\text{wobei: } out = out_o[\,{}^{out_i}/_{\,outer}]$$

Die Ausgabe der resultierenden Projektion erhält dabei die Ausgabe der ursprünglichen äußeren Projektion, wobei darin die äußere Bindungsvariable *outer* durch out_i ersetzt wird.

6 Anwendungsbeispiel

Folgendes Beispiel zeigt die Abbildung einer XQuery-Anfrage auf einen BSA-Algebra-Ausdruck, dessen Umformung und die anschließende Rückabbildung auf eine XQuery-Anfrage. Die XQuery-Anfrage

```
for $b in doc("sample.xml")//Buch,
    $a in (for $x in doc("sample.xml")//Autor
           where $x/text() > "M"
           return $x)
where $a/.. = $b and $b/Titel/text() < "N"
return <Tuple> { $b/Titel, $a } </Tuple>
```

wird zunächst in einen BSA-Ausdruck umgeformt.

$$\Re^{Tuple\langle b/Title::ele,a::ele\rangle::ele}(\Delta_{b/Titel/text()<''N''}(\chi_{a/..=b}(b :: ele \otimes seq_b, a :: ele \otimes seq_a)))$$

$$\begin{aligned} seq_b &:= doc(''sample.xml'')//Buch \\ seq_a &:= \Psi^{x::ele}(\Delta_{x/text()>''M''}(x :: ele \otimes doc(''sample.xml'')//Autor)) \end{aligned}$$

Nun wird die geschachtelte Projektion aufgelöst, indem die Selektionsbedingung in die äußere Operation übernommen und die gebundene Variable der äußeren durch die der inneren Operation ersetzt werden.

$$\Re^{Tuple\langle b/Title::ele,a::ele\rangle::ele}(\Delta_{b/Titel/text()<''N'' \text{ and } x/text()>''M''}(\chi_{a/..=b}(b :: ele \otimes seq_b, a :: ele \otimes seq_a)))$$

$$seq_b := doc(''sample.xml'')//Buch, \quad seq_a := doc(''sample.xml'')//Autor$$

Der in der Verbundbedingung beschriebene Zusammenhang kann nun in die Eingabekomponente übernommen werden.

$$\Re^{Tuple\langle b/Title::ele,a::ele\rangle::ele}(\Delta_{b/Titel/text()<''N'' \text{ and } x/text()>''M''}(\chi_{true}(b :: ele \otimes seq_b, a :: ele \otimes seq_a)))$$

$$seq_b := doc(''sample.xml'')//Buch, \quad seq_a := b/Autor$$

Die von diesem BSA-Ausdruck repräsentierte XQuery-Anfrage sieht damit wie folgt aus:

```
for $b in doc("sample.xml")//Buch,
    $a b/Autor
where $b/Titel/text() < "N" and $a/text() > "M"
return <Tuple> { $b/Titel, $a } </Tuple>
```

7 Ausblick

Die BSA-Algebra befindet sich zur Zeit noch in der Entwicklung. Dabei wird auch die Umsetzung anderer XQuery-Hauptausdrücke berücksichtigt. Für eine frühere Version von BSA existiert in [6] eine Implementierung. [6] beinhaltet ebenfalls eine arbiträre Teilmenge möglicher Optimierungsregeln. Ein vollständiger Regelsatz ist geplant, steht zur Zeit aber noch nicht zur Verfügung. Ebenfalls Gegenstand weiterer Forschung sind die Entwicklung von physischen Operatoren und Kostenmodellen.

Literatur

[1] Ammar Balouch. BSA-Algebra Beta Sequenz Algebra für XQuery. *Graduiertenseminar, Wintersemester 2005/2006, Universität Rostock*, 2005.

[2] Ammar Balouch. XQuery-Optimierung bei Konfliktauflösungsstrategien in der XML-Schemaintegration und bei Zerlegungsstrategien. *Herbstworkshop des Graduiertenkollegs, Universität Rostock*, 2005.

[3] Ammar Balouch, Andreas Heuer, and Holger J. Meyer. BSA, XML Algebra - Operatoren, Regeln und Anwendungen. 2005.

[4] Andreas Heuer and Gunter Saake. *Datenbanken: Implementierungstechniken.* MITP, 1999.

[5] Meike Klettke and Holger Meyer. *XML & Datenbanken: Konzepte, Sprachen, Systeme.* dpunkt.verlag, 2003.

[6] Alf-Christian Schering. X2BCO - Umsetzung von XQuery in BSA und Optimierung. Master's thesis, University of Rostock, 2005.

[7] W3C XML Query Working Group. XQuery 1.0: An XML Query Language - W3C Working Draft 11 February 2005. *http://www.w3.org/TR/2005/WD-xquery-20050211/*, 2005.