Auswertung von XPath-Ausdrücken

Annahme: strukturorientierte Speicherung von XML-Dokumenten – bzw. Varianten davon.

Anfragen nach den Nachfolgern oder Vorgängern eines Knotens eines Baumes können in einem geschlossenen SQL-Ausdruck im Allgemeinen nur mittels Rekursion ausgedrückt werden.

- Anfragen dieser Art können mit einer *festen* Anzahl Verbundoperationen ausgewertet werden,
- wenn wir etwas mehr Informationen über die einzelnen Knoten vorsehen.

Enthaltensein-Anfragen

- Jeder Knoten eines XML-Baumes identifiziert einen Teilbaum und damit einen zusammenhängenden Ausschnitt des Dokumentes.
- Die zugehörigen Ausschnitte des Dokumentes sind ebenso in einander enthalten.
- XPath-Ausdrücke sind in diesem Sinn Enthaltensein-Anfragen, da mittels ihnen eine Menge von Knoten innerhalb eines XML-Baumes, bzw. eines Teilbaumes eines XML-Baumes, lokalisiert werden.
- Ziel ist die Auswertung der `descendant`-, `ancestor`-, `following`- und `preceding`-Achse.

Verfahren XRel

- Idee 1: Den einzelnen Knoten eines XML-Baumes werden Intervalle so zugeordnet, dass direkte und indirekte Vorgänger und Nachfolgerbeziehungen durch Analyse der Intervalle ohne Rekursion entschieden werden können.
- Idee 2: Des Weiteren werden die Pattern-Matching-Möglichkeiten von SQL geschickt ausgenutzt, so dass Lokationspfade, die keine Prädikate enthalten, ohne Verbundoperationen ausgewertet werden können.

XREL ist beschränkt auf XPath-Ausdrücke, die als Achsen lediglich die `child`, `descendant` und `attribute`-Achse enthalten.

benötigte Tabellen

- Für Element-, Attribut- und Textknoten werden jeweils eigene Tabellen verwendet.
- Zusätzlich werden alle über dem XML-Baum ausdrückbaren *einfachen* XPath-Ausdrücke, die ausgehend von der Wurzel ein Blatt lokalisieren, in einer eigenen weiteren Tabelle gespeichert.

  Ein *einfacher* XPath-Ausdruck ist eine Folge von Ausdrücken der Form `#/EName` und `#/@Attr`, wobei '#' ein für das spätere Pattern-Matching benötigtes Trennzeichen ist.

Element

Start	End	PId
0	480	1
9	470	2
25	46	4
47	233	5
56	75	6
76	94	7
95	158	8
102	124	9
125	150	10
159	223	8
166	189	9
190	215	10
234	356	5
243	263	6
264	283	7
284	346	8
291	311	9
312	338	10
357	420	11
363	391	12
392	413	13
421	462	14

Attribut

PId	Start	End	Wert
3	10	10	D
15	422	422	Europe
16	422	422	member

Text

Start	End	Wert	PId
32	38	Germany	4
63	67	Baden	6
84	85	15	7
109	116	Freiburg	9
136	138	198	10
173	181	Karlsruhe	9
201	203	277	10
250	255	Berlin	6
272	274	0,9	7
298	303	Berlin	9
323	326	3472	10
374	379	Europe	12
401	403	100	13

Pfad

PId	PathExpr
1	#/Mondial
2	#/Mondial#/Land
3	#/Mondial#/Land#/@LCode
4	#/Mondial#/Land#/LName
5	#/Mondial#/Land#/Provinz
6	#/Mondial#/Land#/Provinz#/PName
7	#/Mondial#/Land#/Provinz#/Fläche
8	#/Mondial#/Land#/Provinz#/Stadt
9	#/Mondial#/Land#/Provinz#/Stadt#/SName
10	#/Mondial#/Land#/Provinz#/Stadt#/Einwohner
11	#/Mondial#/Land#/Lage
12	#/Mondial#/Land#/Lage#/Kontinent
13	#/Mondial#/Land#/Lage#/Prozent
14	#/Mondial#/Land#/Mitglied
15	#/Mondial#/Land#/Mitglied#/@Organisation
16	#/Mondial#/Land#/Mitglied#/@Art

Jeder Pfad `PathExpr` lokalisiert eine Menge Teilbäumen, bzw. Blattknoten eines erweiterten XML-Baumes.

- Die *Region* eines Text- und Elementknotens ist ein Intervall der Form $[a, b]$, wobei a die Start- und b die Endposition der zugehörigen Tags im Dokument ist.
- Die Region eines Attributknotens ist gegeben durch zwei identische Zahlen, die gleich der Startposition seines Elterelementes erhöht um 1 sind. Hat ein Element mehrere Attribute, so ist in dieser Weise keine Ordnung auf den Attributen impliziert.
- Um die Zuordnung der Element-, Attribut- und Textknoten in den Tabellen zu ihren Positionen im Dokument zu rekonstruieren, sind lediglich die *relativen* Verhältnisse der Werte zu `Start` von Bedeutung.

- /Mondial/Land/Provinz/Stadt:

```
SELECT E.Start, E.End
FROM Element E, Pfad P
WHERE P.PathExp LIKE '#/Mondial#/Land#/Provinz#/Stadt' AND
    E.PId = P.PId
```

- /Mondial//Stadt:

```
SELECT E.Start, E.End
FROM Element E, Pfad P
WHERE P.PathExp LIKE '#/Mondial#%/Stadt' AND
    E.PId = P.PId
```

/Mondial//Provinz//SName:

```
SELECT E.Start, E.End
FROM Element E, Pfad P
WHERE P.PathExp LIKE '#/Mondial#%/Provinz#%/SName' AND
   E.PId = P.PId;
```

```
//Land[Lage/Kontinent = "Europe"]//Stadt/SName:

SELECT E2.Start, E2.End
FROM Pfad P1, Pfad P2, Pfad P3,
    Element E1, Element E2, Text T
WHERE P1.PathExp LIKE '#%/Land'
AND P2.PathExp LIKE '#%/Land#/Lage#/Kontinent'
AND P3.PathExp LIKE '#%/Land#%/Stadt#/SName'
AND E1.PID = P1.PID
AND E2.PID = P3.PID
AND T.PID = P2.PID
AND E1.Start < T.Start AND E1.End > T.End
AND E1.Start < E2.Start AND E1.End > E2.End
AND T.Wert = 'Europe'
```

```
//Land[Lage/Kontinent = "Europe"]//Stadt/SName:

SELECT E2.Start, E2.End
FROM Pfad P1, Pfad P2, Pfad P3,
    Element E1, Element E2, Text T
WHERE P1.PathExp LIKE '#%/Land'
AND P2.PathExp LIKE '#%/Land#/Lage#/Kontinent'
AND P3.PathExp LIKE '#%/Land#%/Stadt#/SName'

--- kein Zusammenhang zwischen P1, P2, P3 ausgedrückt!

AND E1.PID = P1.PID
AND E2.PID = P3.PID
AND T.PID = P2.PID

--- zu P1, P2, P3 gehörende Elemente und Texte bestimmt.

AND E1.Start < T.Start AND E1.End > T.End

--- Land liegt in Europa.

AND E1.Start < E2.Start AND E1.End > E2.End

--- Name einer europäischen Stadt.

AND T.Wert = 'Europe'
```

//Stadt[SName = //Provinz[PName = "Baden"]//SName]:

```
SELECT E1.S, E1.E
FROM XRelPfad P1, XRelPfad P2, XRelPfad P3, XRelPfad P4, XRelPfad P5,
    XRelElement E1, XRelElement E2, XRelElement E3,
    XRelElement E4, XRelElement E5,
    XRelText T1, XRelText T2, XRelText T3
WHERE P1.PathExpr LIKE '#%/Stadt'
AND P2.PathExpr LIKE '#%/Stadt#/SName'
AND P3.PathExpr LIKE '#%/Provinz'
AND P4.PathExpr LIKE '#%/Provinz#/PName'
AND P5.PathExpr LIKE '#%/Provinz#%/SName'
AND E1.PId = P1.PId AND E2.PId = P2.PId AND E3.PId = P3.PId
AND E4.PId = P4.PId AND E5.PId = P5.PId
AND E4.S < T1.S AND E4.E > T1.E AND T1.Wert = 'Baden'
AND E2.S < T2.S AND E2.E > T2.E
AND E5.S < T3.S AND E5.E > T3.E
AND T2.Wert = T3.Wert
AND E1.S < E2.S AND E1.E > E2.E
AND E3.S < E4.S AND E3.E > E4.E
AND E3.S < E5.S AND E3.E > E5.E
```

//Stadt[SName = //Provinz[PName = "Baden"]//SName]:

```
SELECT E1.S, E1.E
FROM XRelPfad P1, XRelPfad P2, XRelPfad P3, XRelPfad P4, XRelPfad P5,
    XRelElement E1, XRelElement E2, XRelElement E3,
    XRelElement E4, XRelElement E5,
    XRelText T1, XRelText T2, XRelText T3
WHERE P1.PathExpr LIKE '#%/Stadt'
AND P2.PathExpr LIKE '#%/Stadt#/SName'
AND P3.PathExpr LIKE '#%/Provinz'
AND P4.PathExpr LIKE '#%/Provinz#/PName'
AND P5.PathExpr LIKE '#%/Provinz#%/SName'
--- kein Zusammenhang zwischen P1, ..., P5 ausgedrückt!
AND E1.PId = P1.PId AND E2.PId = P2.PId AND E3.PId = P3.PId
AND E4.PId = P4.PId AND E5.PId = P5.PId
--- zugehörende Elemente bestimmt.
AND E4.S < T1.S AND E4.E > T1.E AND T1.Wert = 'Baden'
--- Provinzname 'Baden' festgeschrieben.
AND E3.S < E4.S AND E3.E > E4.E
--- Provinz Baden.
AND E3.S < E5.S AND E3.E > E5.E
--- Name einer badischen Stadt.
AND E2.S < T2.S AND E2.E > T2.E
AND E5.S < T3.S AND E5.E > T3.E
AND T2.Wert = T3.Wert
--- Stadtname identisch zu einem Namen einer badischen Stadt.
AND E1.S < E2.S AND E1.E > E2.E
--- Stadt mit einem Namen, der identisch zu einem Namen einer badischen Stadt ist.
```

Haupt-/Nebenrang-Darstellung

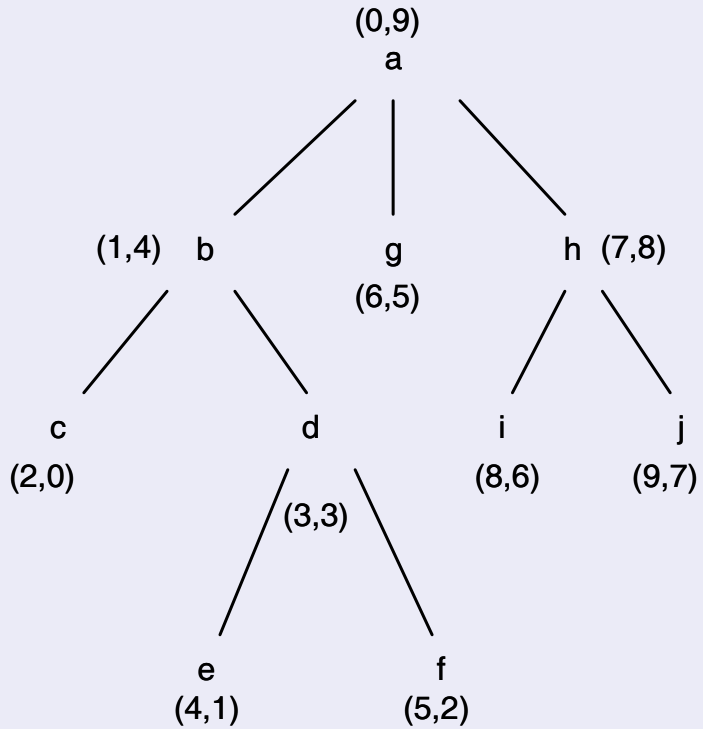
- Wir ordnen jedem Knoten v eines XML-Baumes T seinen *Hauptrang* (engl. *preorder rank*) $pre(v)$ und seinen *Nebenrang* (engl. *postorder rank*) $post(v)$ zu.
- Haupt-, bzw. Nebenrang der einzelnen Knoten ergeben sich aus ihrer Position bzgl. der Haupt-, bzw. Nebenreihenfolge innerhalb des XML-Baumes.
- Die Hauptreihenfolge ergibt sich, indem wir, beginnend mit der Wurzel, einen Knoten des Baumes *vor* seinen Kindknoten betrachten und die Kindknoten anschließend rekursiv in der Reihenfolge von links nach rechts.
- Die Nebenreihenfolge ergibt sich, indem wir, wiederum beginnend mit der Wurzel, einen Knoten betrachten, *nachdem* wir rekursiv seine Kindknoten von links nach rechts betrachtet haben.

Rangberechnung nach Haupt-/Nebenreihenfolge

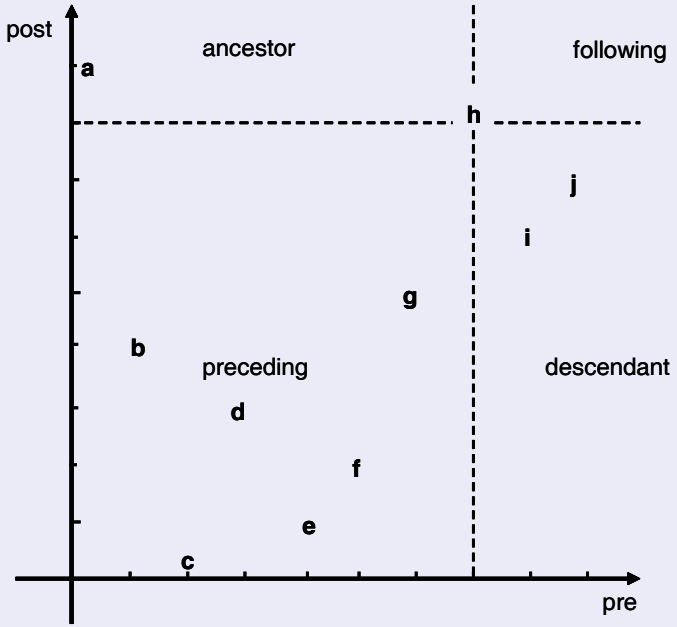
```
function Rang(node) {
    var children = node.childNodes;
    alert('Hauptrang zu '+node.id+': '+hauptRang);
    hauptRang++;
    for (var i = 0; i < children.length; i++)
       Rang(children[i]);
    alert('Nebenrang zu '+node.id+': '+nebenRang);
    nebenRang++;
}
```

Beispiel

```
<a>
  <b>
    <c>  </c>
    <d>
      <e>  </e>
      <f>  </f>
    </d>
  </b>
  <g>  </g>
  <h>
    <i>  </i>
    <j>  </j>
  </h>
</a>
```



Koordinaten-Darstellung



v' ist Element der `descendant`-Achse von $v \iff$
$pre(v) < pre(v')$ und $post(v) > post(v')$,

v' ist Element der `ancestor`-Achse von $v \iff$
$pre(v) > pre(v')$ und $post(v) < post(v')$,

v' ist Element der `preceding`-Achse von $v \iff$
$pre(v) > pre(v')$ und $post(v) > post(v')$,

v' ist Element der `following`-Achse von $v \iff$
$pre(v) < pre(v')$ und $post(v) < post(v')$.

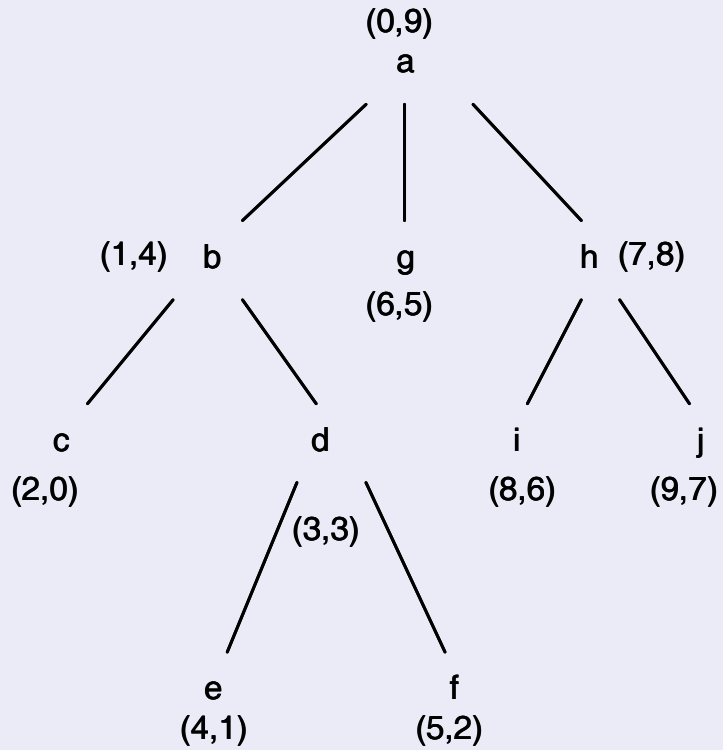
Haupt-/Nebenrang: Relationale Darstellung

- Im Folgenden ausschließlich Elementknoten eines XML-Baumes.
- `Element(Pre, Post, Name)`; Haupt- und Nebenrang und Elementnamen.
- `Pre` ist der Schlüssel.

Beispiel: d/descendant::*

Element

Pre	Post	Name
0	9	a
1	4	b
2	0	c
3	3	d
4	1	e
5	2	f
6	5	g
7	8	h
8	6	i
9	7	j



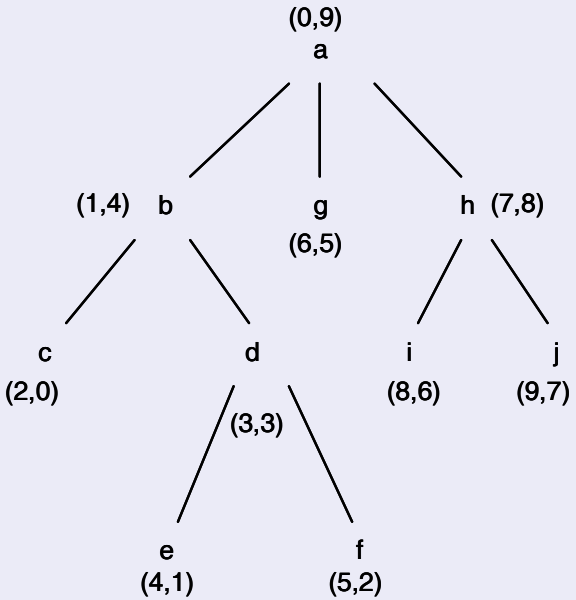

```
SELECT E2.Pre
FROM Element E1, Element E2
WHERE E1.Name = 'd' AND
   E1.Pre < E2.Pre AND E1.Post > E2.Post
```

ein Zusammenhang zwischen Haupt- und Nebenrang

Betrachte einen XML-Baum.

- Sei $size(v)$ die Anzahl Knoten des Teilbaumes mit Wurzel v.
- Sei $level(v)$ die Tiefe des Knotens v.
- Es gilt: $post(v) = pre(v) + size(v) - level(v)$.

zu $post(v) = pre(v) + size(v) - level(v)$



pre	post	size	level	element
0	9	9	0	a
1	4	4	1	b
2	0	0	2	c
3	3	2	2	d
4	1	0	3	e
5	2	0	3	f
6	5	0	1	g
7	8	2	1	h
8	6	0	2	i
9	7	0	2	j

Auswertung

```
SELECT E2.Pre
FROM Element E1, Element E2
WHERE E1.Name = 'd' AND
   E1.Pre < E2.Pre AND E1.Post > E2.Post
```

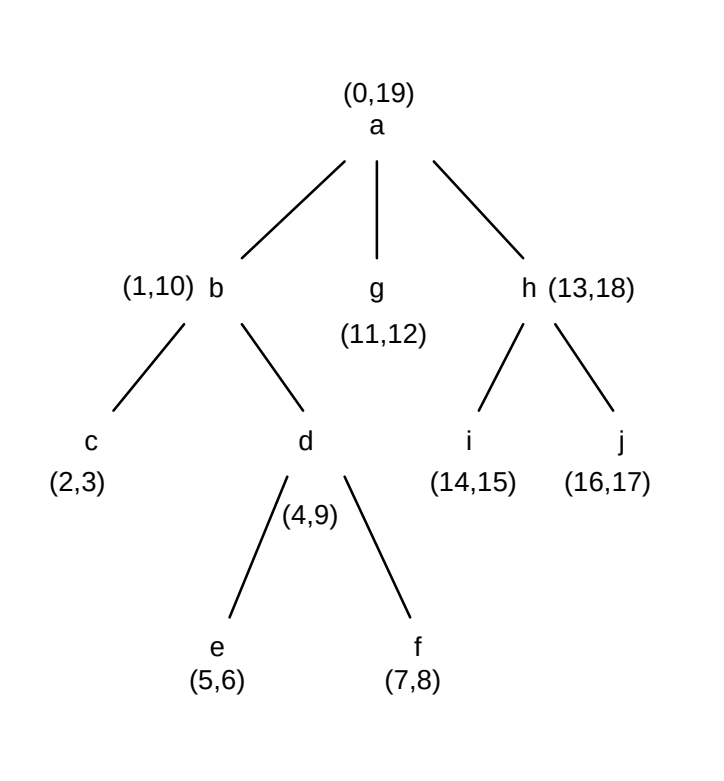
- Bestimme diejenigen Zeilen, die die Bedingung $pre(\mathtt{d}) < pre(v')$ erfüllen,
- Bestimme die Menge der Zeilen, die die Bedingung $post(\mathtt{d}) > post(v')$ erfüllen.
- Berechne den Schnitt beider Mengen.

Bei XML-Bäumen mit einer großen Anzahl Knoten können die als Zwischenergebnis berechneten Mengen erheblichen Umfang annehmen, so dass die Frage nach einer effizienteren Technik relevant wird.

Beachte, dass ein Index über `Pre` und ein Index über `Post` benötigt wird.

Optimierung

Lediglich das *relative* Verhältnis der Rangwerte zueinander ist von Bedeutung und nicht die absoluten Werte.



$\texttt{Element}^{opt}$

$\widetilde{\texttt{Pre}}$	$\widetilde{\texttt{Post}}$	Name
0	19	a
1	10	b
2	3	c
4	9	d
5	6	e
7	8	f
11	12	g
13	18	h
14	15	i
16	17	j

Bemerkung: XREL läßt grüßen!?

Rangberechnung für Optimierung

```
function Rang(node) {
    var children = node.childNodes;
    alert('Hauptrang zu '+node.id+': '+RangZähler);
    RangZähler++;
    for (var i = 0; i < children.length; i++)
       Rang(children[i]);
    alert('Nebenrang zu '+node.id+': '+RangZähler);
    RangZähler++;
}
```

Descendant-Achse optimiert

v' ist Element der `descendant`-Achse von v genau dann, wenn

$$\tilde{\texttt{Pre}}(v) < \tilde{\texttt{Pre}}(v') < \tilde{\texttt{Post}}(v), \text{ bzw., } \tilde{\texttt{Pre}}(v) < \tilde{\texttt{Post}}(v') < \tilde{\texttt{Post}}(v).$$

Beachte, $\tilde{\texttt{Pre}}(v)$ und $\tilde{\texttt{Post}}(v)$ sind hier Konstante. Somit wird nur entweder ein Index über `Pre` oder über `Post` benötigt.
Für die übrigen Achsen kann eine vergleichbare Optimierung nicht erreicht werden – die `ancestor` v' eines Knotens v können zwar wie folgt definiert werden: $\tilde{\texttt{Pre}}(v') < \tilde{\texttt{Pre}}(v) < \tilde{\texttt{Post}}(v')$, hier benötigen wir jedoch wiederum einen Index über `Pre` und `Post`.

Beispiel `descendant(d)`

$\tilde{\texttt{Pre}}$(d)	$\tilde{\texttt{Post}}$(d)	$\tilde{\texttt{Pre}}$	$\tilde{\texttt{Post}}$	Name
4	9	0	19	a
4	9	1	10	b
4	9	2	3	c
4	9	4	9	d
4	9	5	6	e
4	9	7	8	f
4	9	11	12	g
4	9	13	18	h
4	9	14	15	i
4	9	16	17	j

optimierte XPath-Auswertung in SQL

(1) `/descendant::B/descendant::*`.

```
SELECT E3.Pre
FROM Element E1, Element E2, Element E3
WHERE E1.Pre = 0 AND E2.Name = 'B' AND
   E1.Pre < E2.Pre AND E2.Pre < E1.Post AND
   E2.Pre < E3.Pre and E3.Pre < E2.Post
```

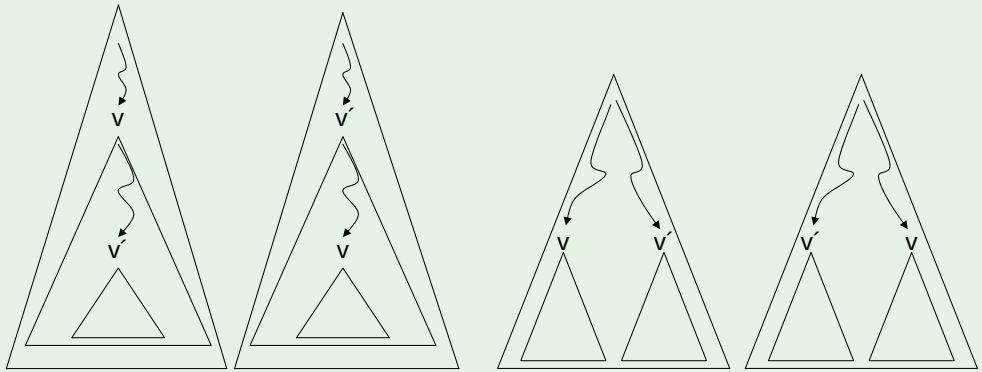
(2) `/descendant::B/ancestor::A`.

```
SELECT E3.Pre
FROM Element E1, Element E2, Element E3
WHERE E1.Pre = 0 AND E2.Name = 'B' AND E3.Name = 'A' AND
   E1.Pre < E2.Pre AND E2.Pre < E1.Post AND
   E2.Pre > E3.Pre AND E2.Post < E3.Post
```

(3) `/descendant::A[descendant::B]` – äquivalent zu (2), jedoch ohne Rückwärtsachse und somit optimierbar!

```
SELECT E2.Pre
FROM Element E1, Element E2, Element E3
WHERE E1.Pre = 0 AND E2.Name = 'A' AND E3.Name = 'B' AND
   E1.Pre < E2.Pre AND E2.Pre < E1.Post AND
   E2.Pre < E3.Pre AND E3.Pre < E2.Post
```

weitere Achsencharakterisierungen als Grundlage für Optimierungen

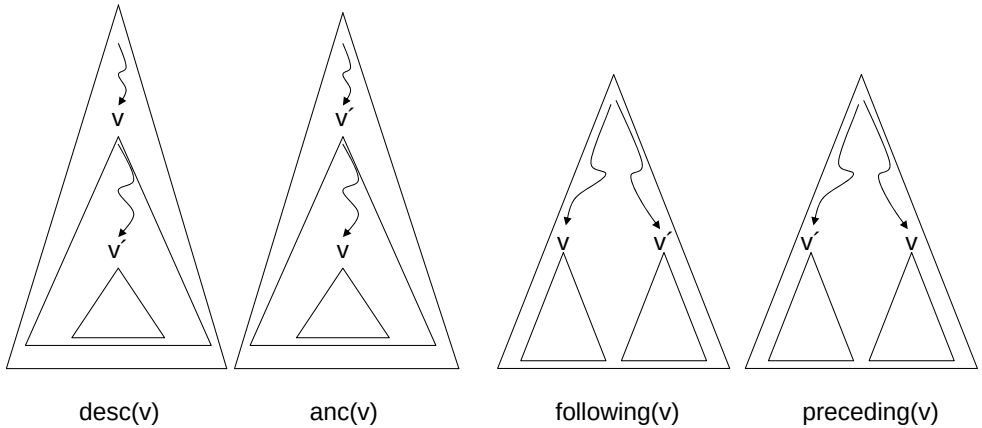


v' ist Element der `descendant`-Achse von $v \iff$
$pre(v) < pre(v')$ und $pre(v') \leq pre(v) + size(v)$,

v' ist Element der `ancestor`-Achse von $v \iff$
$pre(v) > pre(v')$ und $pre(v) \leq pre(v') + size(v')$,

v' ist Element der `preceding`-Achse von $v \iff pre(v') + size(v') < pre(v)$,

v' ist Element der `following`-Achse von $v \iff pre(v') > pre(v) + size(v)$.



v' ist Element der `ancestor`-Achse von $v \iff pre(v) > pre(v')$ und $pre(v) \leq pre(v') + size(v')$.

(1) Wegen $pre(v) > pre(v')$ kann v' kein Following sein, also höchstens Anchestor oder Preceding.

(2) Falls v' Anchestor zu v, gilt $pre(v') + size(v') \geq pre(v)$ wie gefordert.

Angenommen v' Preceding zu v.
Es gilt $pre(v') + size(v') < pre(v'')$ für jeden beliebigen Anchestor v'' von v.
Also folgt wegen (1):

$$pre(v') + size(v') < pre(v'') < pre(v),$$

somit ein Widerspruch.

weitere Achsen

v' ist Element der `self`-Achse von $v \iff pre(v') = pre(v)$,

v' ist Element der `descendant-or-self`-Achse von $v \iff$
$pre(v) \leq pre(v')$ und $pre(v') \leq pre(v) + size(v)$,

v' ist Element der `ancestor-or-self`-Achse von $v \iff$
$pre(v) \geq pre(v')$ und $pre(v) < pre(v') + size(v')$,

v' ist Element der `child`-Achse von $v \iff level(v') = level(v) + 1$ und
$pre(v) < pre(v')$, $pre(v') \leq pre(v) + size(v)$,

v' ist Element der `parent`-Achse von $v \iff level(v) = level(v') + 1$ und
$pre(v) > pre(v')$, $pre(v) \leq pre(v') + size(v')$,

v' ist Element der `following-sibling`-Achse von $v \iff$
$pre(v') > pre(v) + size(v)$ und es existiert ein v'' so,
dass v'' sowohl Element der `parent`-Achse von v als auch von v',

v' ist Element der `preceding-sibling`-Achse von $v \iff$
$pre(v') + size(v') < pre(v)$ und es existiert ein v'' so,
dass v'' sowohl Element der `parent`-Achse von v als auch von v'.

XRel und XPath-Accelerator

- *XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases*, M. Yoshikawa et al., ACM ToIT, Vol. 1, No. 1, 2001.
- *Acccelerating XPath Evaluation in Any RDBMS*, T. Grust et al., ACM ToDS, Vol. 29, No 1, 2004.
- Die grundlegenden Ideen der beiden Ansätze können kombiniert werden.

Relational XML

Effiziente Auswertung von XPath mittels SQL

- direkte Umsetzung in SQL,
- zusätzliche Restriktionen,
- Vermeiden von Duplikaten.

Literatur: *Staircase Join: Teach a Relational DBMS to Watch its (Axis) Steps*, Torsten Grust, Maurice van Keulen, Jens Teubner, in Proc. VLDB 2003.

SQL mit zusätzlichen Restriktionen

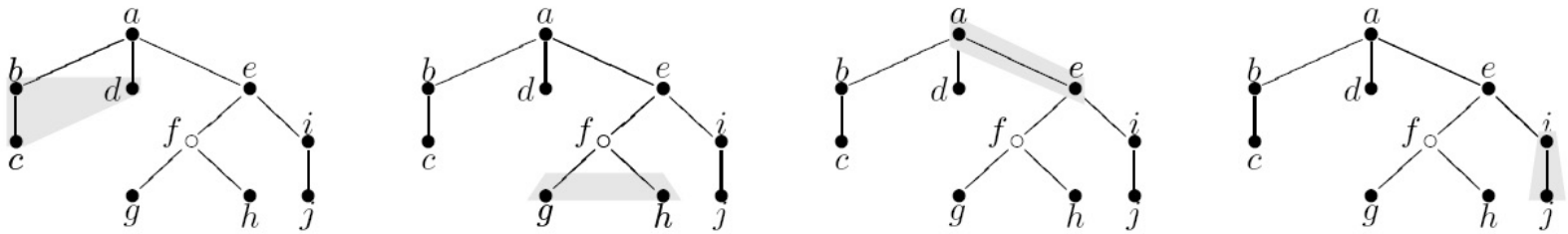
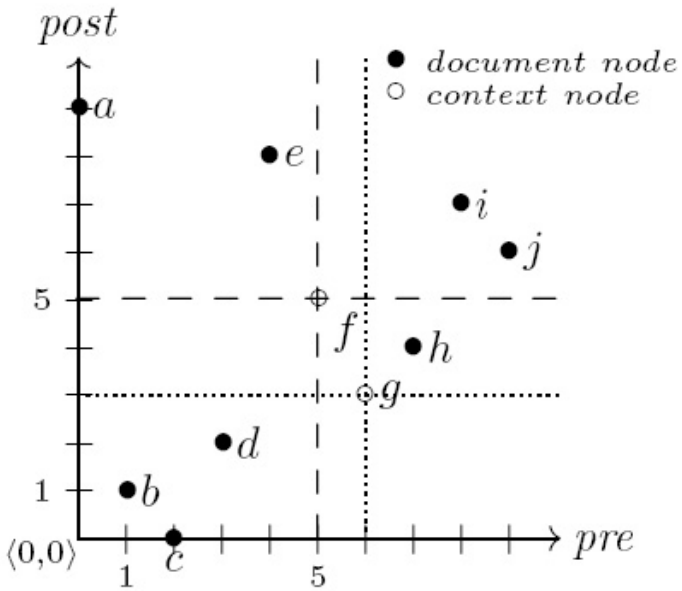


Figure 1: XPath axes induce document regions: shaded nodes are reachable from context node f via a step along the (a) `preceding`, (b) `descendant`, (c) `ancestor`, (d) `following` axes. Leaf nodes denote either empty XML elements, attributes, text, comment, or processing instruction nodes; inner nodes represent non-empty elements.



	pre	*post*
a	0	9
b	1	1
c	2	0
d	3	2
e	4	8
f	5	5
g	6	3
h	7	4
i	8	7
j	9	6

```
1 SELECT DISTINCT v2.pre
2   FROM doc v1, doc v2
3  WHERE v1.pre > pre(c)
4    AND v1.pre < v2.pre
5    AND v1.post > post(c)
6    AND v1.post > v2.post
8  ORDER BY v2.pre
```

(c)/following/descendant $= (f, g, h, i, j)$.

zusätzliche Abschätzung

7 AND $v_2.pre \leq v_1.post + h$ AND $v_2.post \geq v_1.pre + h$

$$\begin{aligned} |\,(v)/descendant\,| &= post(v) - pre(v) + level(v) \\ &\leq post(v) - pre(v) + h \end{aligned}$$

- $v_1.pre < v_2.pre$
  $v_1.pre < v_1.pre + v_1.post - v_1.pre + h$
  $v_2.pre \leq v_1.post + h$
- $v_1.post > v_2.post$
  $v_2.post > v_1.post - v_1.post + v_1.pre - h$
  $v_2.post \geq v_1.pre - h$

Beispiel: Vermeiden von (einigen) Duplikaten ancestor-or-self

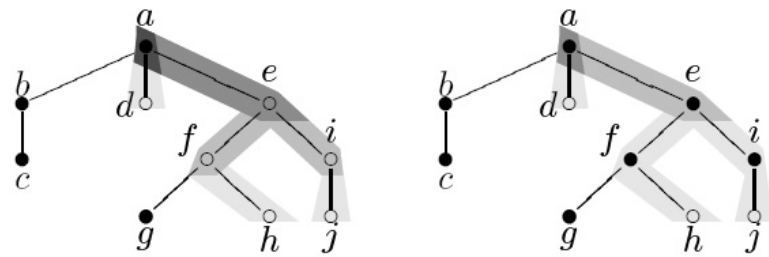


Figure 4: (a) Intersection and inclusion of the `ancestor-or-self` paths of a context node sequence. (b) The pruned context node sequence covers the same `ancestor-or-self` region and produces less duplicates (3 rather than 11).

$$(d, e, f, h, i, j)/ancestor - or - self = (d, h, j)/ancestor - or - self$$

Vermeiden von Duplikaten: allgemeines Pruning

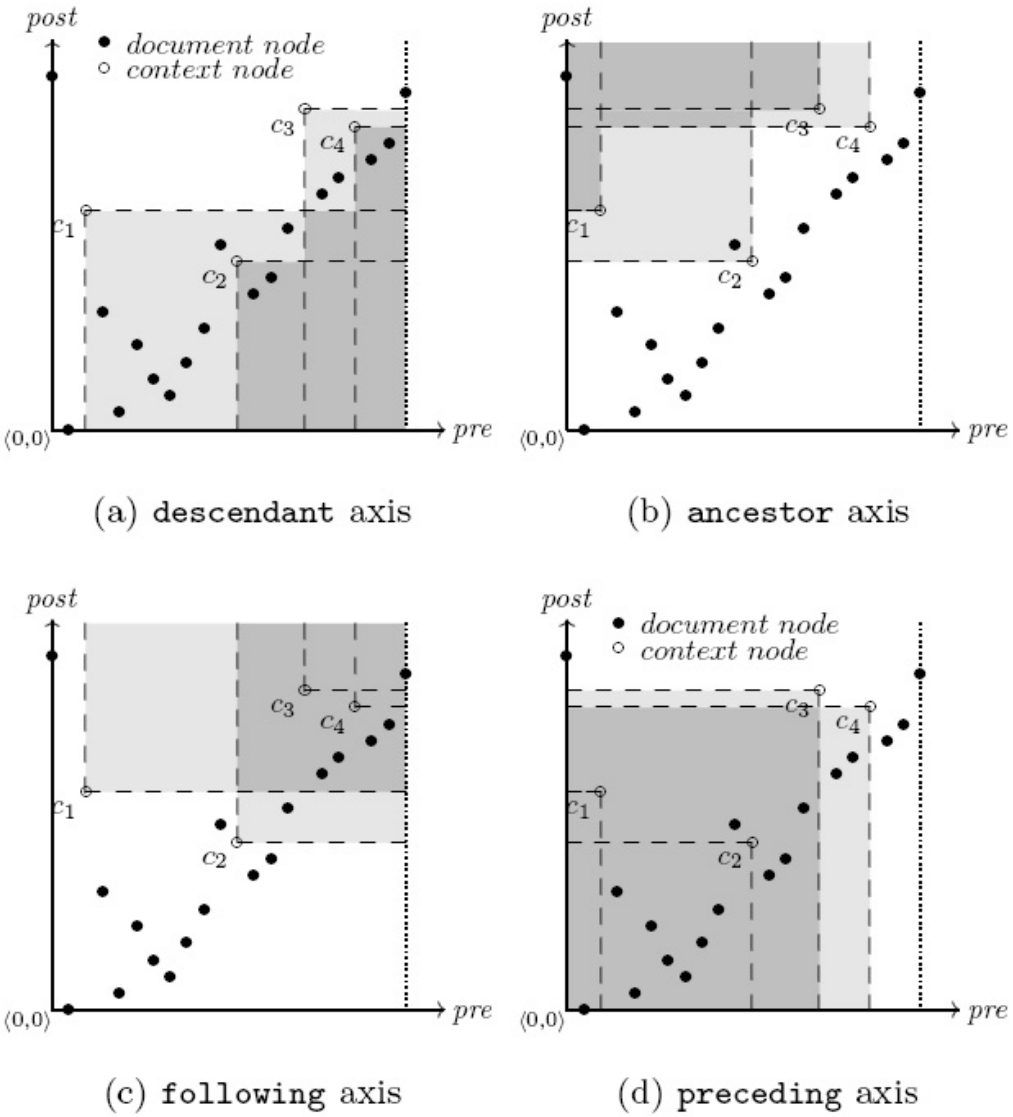


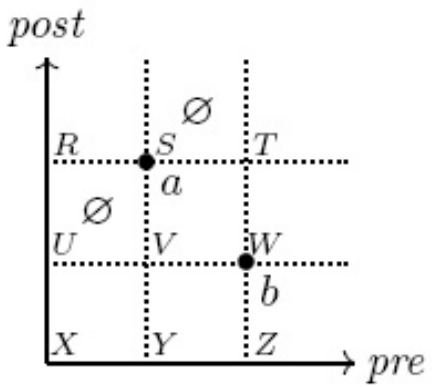
Figure 5: Overlapping regions (context nodes c_i).

$(c_1, c_2, c_3, c_4)/descendant = (c_1, c_3)/descendant$

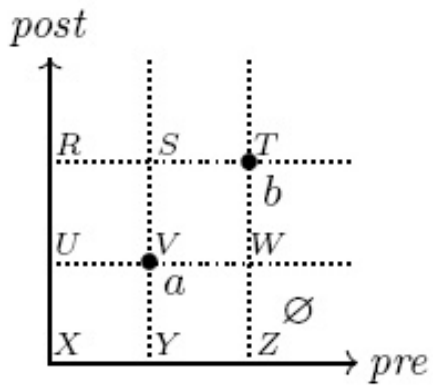
$(c_1, c_2, c_3, c_4)/ancestor = (c_2, c_4)/ancestor$

$(c_1, c_2, c_3, c_4)/following = (c_1, c_2)/following$

$(c_1, c_2, c_3, c_4)/preceding = (c_3, c_4)/preceding$



(a) Nodes a and b relate to each other on the `ancestor/descendant` axis.



(b) Nodes a and b relate to each other on the `preceding/following` axis.

Figure 7: Empty regions in the *pre*/*post* plane.

(a): Treppenkurve nach Pruning der following/preceding-Achse.

$$(a, b)/\mathit{following} = S \cup T \cup W = T \cup W = (b)/\mathit{following}$$

$$(a, b)/\mathit{preceding} = U \cup X \cup Y = X \cup Y = (b)/\mathit{preceding}$$

Es genügt einen einzigen Knoten zu betrachten! Bzgl. following-Achse, der Knoten mit minimalem postorder-Wert; bzgl. preceding-Achse, der Knoten mit maximalem preorder-Wert.
(b): Treppenkurve nach Pruning der descending/anchestor-Achse.

Anchestor/Descending-Achse: Partitionierung

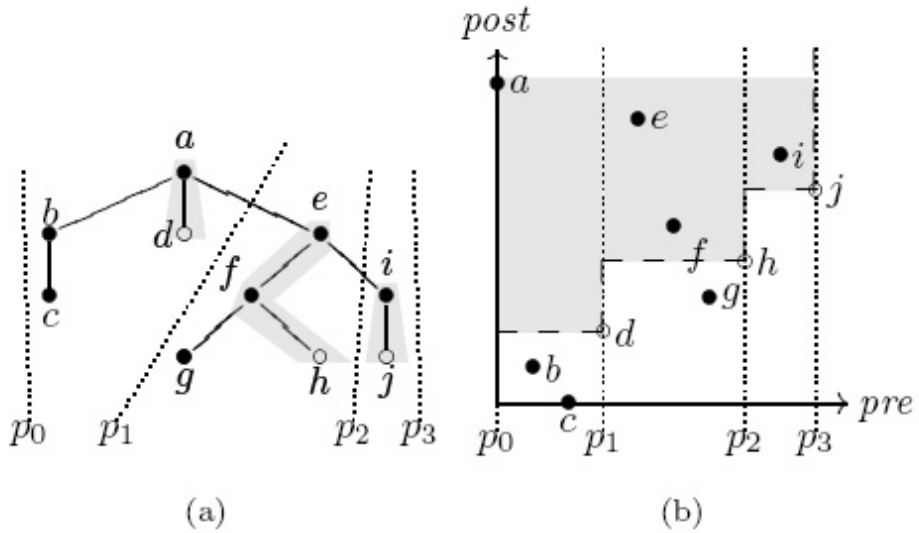


Figure 8: The partitions $[p_0, p_1)$, $[p_1, p_2)$, $[p_2, p_3)$ of the `ancestor` staircase separate the `ancestor-or-self` paths in the document tree.

Staircase Join

- Pruning (on the fly) + Partitionierung
- Tabellen werden einmal durchlaufen.
- Es werden keine Duplikate erzeugt.

Anchestor/Descending-Achse: Skipping

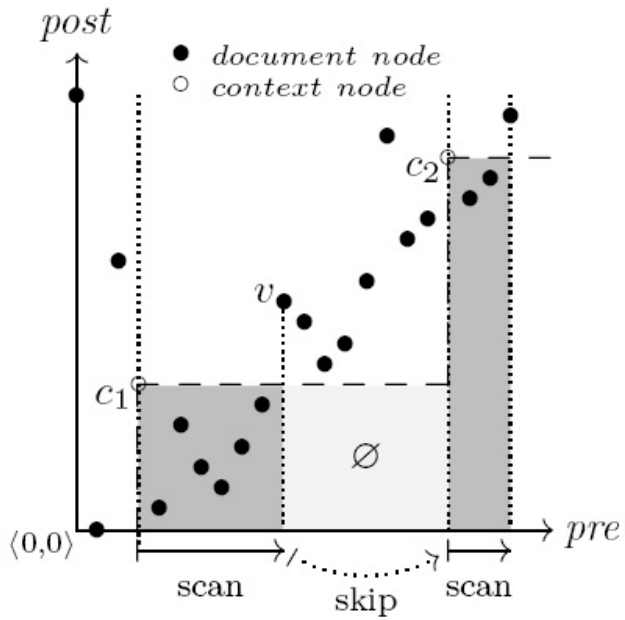


Figure 9: Skipping technique for `descendant` axis.

Staircase Join weiter optimiert

- Die Z-Region der Treppenkurve bei Evaluierung der Descendant/Anchestor-Achse ist leer.
- Z-Regionen können übersprungen werden.

Staircase Join

Pruning descendant-Achse

```
prunecontext_desc (context : TABLE (pre,post)) ≡
  BEGIN
    result ← NEW TABLE (pre, post); prev ← 0;
    FOREACH c IN context DO
      IF c.post > prev THEN
        APPEND c TO result;
        prev ← c.post;
    RETURN result;
  END
```

Algorithm 1: Context pruning for `descendant` axis, table context is assumed to be *pre*-sorted.

Staicase Join descendant-Achse

```
staircasejoin_desc (doc : TABLE (pre,post),
                    context : TABLE (pre,post)) ≡
  BEGIN
    result ← NEW TABLE (pre, post);
    FOREACH SUCCESSIVE PAIR (c1, c2) IN context DO
      scanpartition (c1.pre + 1, c2.pre − 1, c1.post,<);
    c ← LAST NODE IN context;
    n ← LAST NODE IN doc;
    scanpartition (c.pre + 1, n.pre, c.post,<);
    RETURN result;
  END
```

```
scanpartition (pre1 ,pre2 ,post,θ) ≡
  BEGIN
    FOR i FROM pre1 TO pre2 DO
      IF doc[i].post θ post THEN
        APPEND doc[i] TO result;
  END
```

Algorithm 2: Staircase join algorithms (`descendant` and `ancestor` axes).

Skipping descendant-Achse

```
scanpartition_desc (pre1, pre2, post) ≡
  BEGIN
    FOR i FROM pre1 TO pre2 DO
(⋆)   IF doc[i].post < post THEN
        APPEND doc[i] TO result;
      ELSE
        BREAK;    /* skip */
  END
```

Algorithm 3: Skipping for the descendant axis.

Literatur

- Masatoshi Yoshikawa, Toshiyuki Amagasa, Takeyuki Shimura, Shunsuke Uemura: *XRel: a path-based approach to storage and retrieval of XML documents using relational databases.* ACM Trans. Internet Techn. 1(1): 110-141 (2001)
- Torsten Grust, Maurice van Keulen, Jens Teubner: *Accelerating XPath evaluation in any RDBMS.* ACM Trans. Database Syst. 29: 91-131 (2004)
- Paul F. Dietz, Daniel Dominic Sleator: *Two Algorithms for Maintaining Order in a List.* STOC 1987: 365-372
- Joe Celko: *SQL for Smarties: Advanced SQL Programming.* Second Edition Morgan Kaufmann 1999