Semistrukturierte Daten

XPath

Stefan Woltran
Emanuel Sallinger

Institut für Informationssysteme
Technische Universität Wien

Sommersemester 2012

Inhalt

Überblick

Familie von Standards:

- XPath: Navigation in XML Dokumenten
- XQuery: Abfragen auf XML Dokumenten
- XSLT: Transformation von XML Dokumenten

Seit XPath 2.0, XQuery 1.0, XSLT 2.0:

- Gemeinsames Datenmodell
- Gemeinsame Funktionen und Operatoren

XPath 2.0

- XPath ist die Basis für viele XML-related Standards:
  - insbesondere für XQuery und XSLT
  - in eingeschränkter Form auch für XML Schema
  - aber auch für XPointer
  - XPath ist selbst nicht in XML Notation
- Hauptaufgaben:
  - Navigation im Dokumentenbaum
  - Selektion von (Knoten-)Sequenzen
  - Einfache Operationen auf Inhalten
- Versionen:
  - XPath 1.0: Recommendation seit 1999
  - XPath 2.0: Recommendation seit 2007: Fundamentale Änderungen (neues Datenmodell, Nutzung von XML Schema Typen)

Datenmodell

- Überblick
- Knoten
- Document Order
- Sequences
- XPath Auswertung

Überblick

- XPath 2.0 basiert auf dem XQuery 1.0 und XPath 2.0 Data Model (XDM)
- XDM ist das Datenmodell für XPath 2.0, XSLT 2.0 und XQuery 1.0
- XDM unterstützt unter Anderem:
  - XML Schema Typen (Strukturen, simple Datentypen, ...)
  - typisierte atomare Werte
  - Sequenzen
  - Verwendung mehrerer Dokumente
- Das XML-Dokument wird als Baum betrachtet
- Dieser Baum ist leicht abweichend vom "DOM"

Knoten

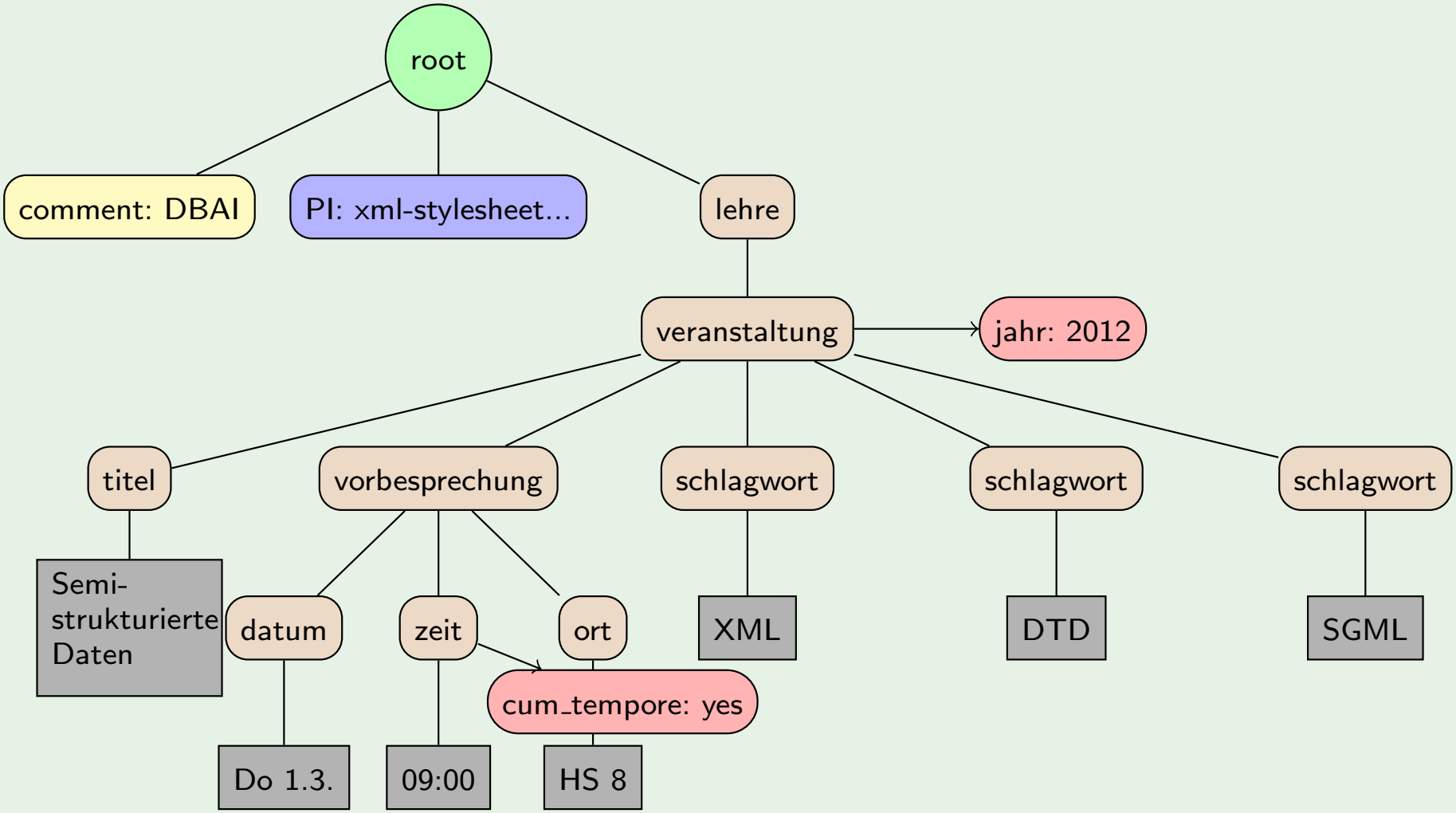
- insgesamt 7 Arten von Knoten im Dokumentenbaum:

Document Node	Wurzelknoten des Baums
Element Node	für jedes Element im Dokument
Attribute Node	assoziiert mit entsprechendem Element Node
Namespace Node	für alle NS-Präfixe plus einen etwaigen Default-NS, die für ein Element gültig sind
Processing Instruction Node	für jede Processing Instruction
Comment Node	für jeden Kommentar
Text Node	Zeichendaten werden in möglichst große Text Nodes zusammengefasst

Beispiel (XML-Dokument)

```
<?xml version='1.0'? encoding='ISO-8859-1'>
<!-- DBAI -->
<?xml-stylesheet type='text/css' href='lehre.css'?>
<lehre>
  <veranstaltung jahr='2012'>
    <titel>Semistrukturierte Daten</titel>
    <vorbesprechung>
      <datum>Do 1.3.</datum>
      <zeit cum_tempore='yes'>09:00</zeit>
      <ort>HS 8</ort>
    </vorbesprechung>
    <schlagwort>XML</schlagwort>
    <schlagwort>DTD</schlagwort>
    <schlagwort>SGML</schlagwort>
  </veranstaltung>
</lehre>
```

Beispiel (Dokumentenbaum)



Knoten

- Knoten können unter Anderem folgende Informationen liefern:

Knotenname qualifizierter Name bei Element und Attribute Nodes, Präfix bei Namespace Nodes, Target bei PIs

Elternknoten jeder Knoten außer der Document Node hat genau einen Elternknoten

Kindknoten nur bei Document Nodes und Element Nodes
(Attributknoten und Namespaceknoten sind keine Kindknoten!)

Attribute nur bei Element Nodes
(Namespace-Deklarationen – `xmlns...` – sind keine Attributknoten!)

Namespaces nur bei Element Nodes
(für jede für das Element gültige Namespace-Bindung)

Typ Typinformation zum Knoten (type-name, typed-value)

String-Wert eines Knotens

Document Node	Konkatenation aller Textknoten des Dokuments
Element Node	Konkatenation aller Textknoten unterhalb eines Elementknotens
Attribute Node	normalisierter Attributwert
Namespace Node	Namespace URI
PI Node	String hinter dem Target der PI, z.B. `type='text/css' href='lehre.css'`
Comment Node	Inhalt des Kommentars, ohne `<!--` und `-->`
Text Node	Zeicheninhalt

Document Order

- alle Knoten im Baum sind geordnet (Totalordnung)
- Ordnung der Element Nodes im Baum ist top-down, left-to-right (d.h.: die Reihenfolge der Start-Tags ist entscheidend)
- Nach einem Elementknoten kommen – in dieser Reihenfolge – dessen
  - Namespace Nodes
  - Attribute Nodes
  - Kindknoten
- Reihenfolge innerhalb der Namespace Nodes eines Elements bzw. innerhalb der Attribute Nodes eines Elements ist implementierungsabhängig
- Ordnung der Knoten im Resultat eines XPath-Ausdrucks:
  - normalerweise in document order
  - bei Navigation in umgekehrter Richtung: reverse document order

Sequences

- ein XPath-Ausdruck liefert als Ergebnis immer eine Sequence
- eine Sequenz besteht aus beliebig vielen Items
- ein Item ist entweder ein Knoten oder ein atomarer Wert (`xs:string`, `xs:boolean`, `xs:decimal`, ...)
- ein Item `x` ist äquivalent zu einer Sequenz, die nur dieses eine Item `x` enthält
- in Sequenzen sind Duplikate erlaubt
- Sequenzen können keine weiteren Sequenzen enthalten, sie sind "flach" z.B.: `(a b (c d))` entspricht `(a b c d)`

XPath Auswertung

- Expression = zentrales syntaktisches Konstrukt in XPath
- Die Auswertung einer XPath Expression geschieht immer relativ zu einem Kontext:

context-node	Knoten im Dokumentenbaum
context-position	positiver Integer
context-size	positiver Integer
variable values	für alle Variablen einer XPath expression (Variablennotation: `$x`)

- context-node/position/size können sich während der Auswertung einer XPath Expression ändern (z.B. innerhalb eines Unterausdrucks)

Pfadangaben (Paths)

- Überblick
- Steps
- Achsen
- Node Tests
- Abkürzungen
- Filter (Predicates)
- Filterlisten
- Auswertung von Steps

Überblick

- wichtigste Form von XPath Expressions: Pfadangaben (= paths)
- ein Pfad besteht aus ein oder mehreren Schritten (= steps)
- Absoluter Pfad:
  - beginnt bei der Document Node
  - Schreibweise: z.B. `/lehre/veranstaltung/schlagwort`
- Relativer Pfad:
  - beginnt beim aktuellen Context Node
  - Schreibweise: z.B. `veranstaltung/schlagwort`

Beispiele

Absoluter Pfad:

```
/lehre/veranstaltung/titel
/child::lehre/child::veranstaltung/child::titel
```

Relativer Pfad:

```
veranstaltung/titel
child::veranstaltung/child::titel

zeit/@cum_tempore
child::zeit/attribute::cum_tempore
```

Verwendung von Filtern:

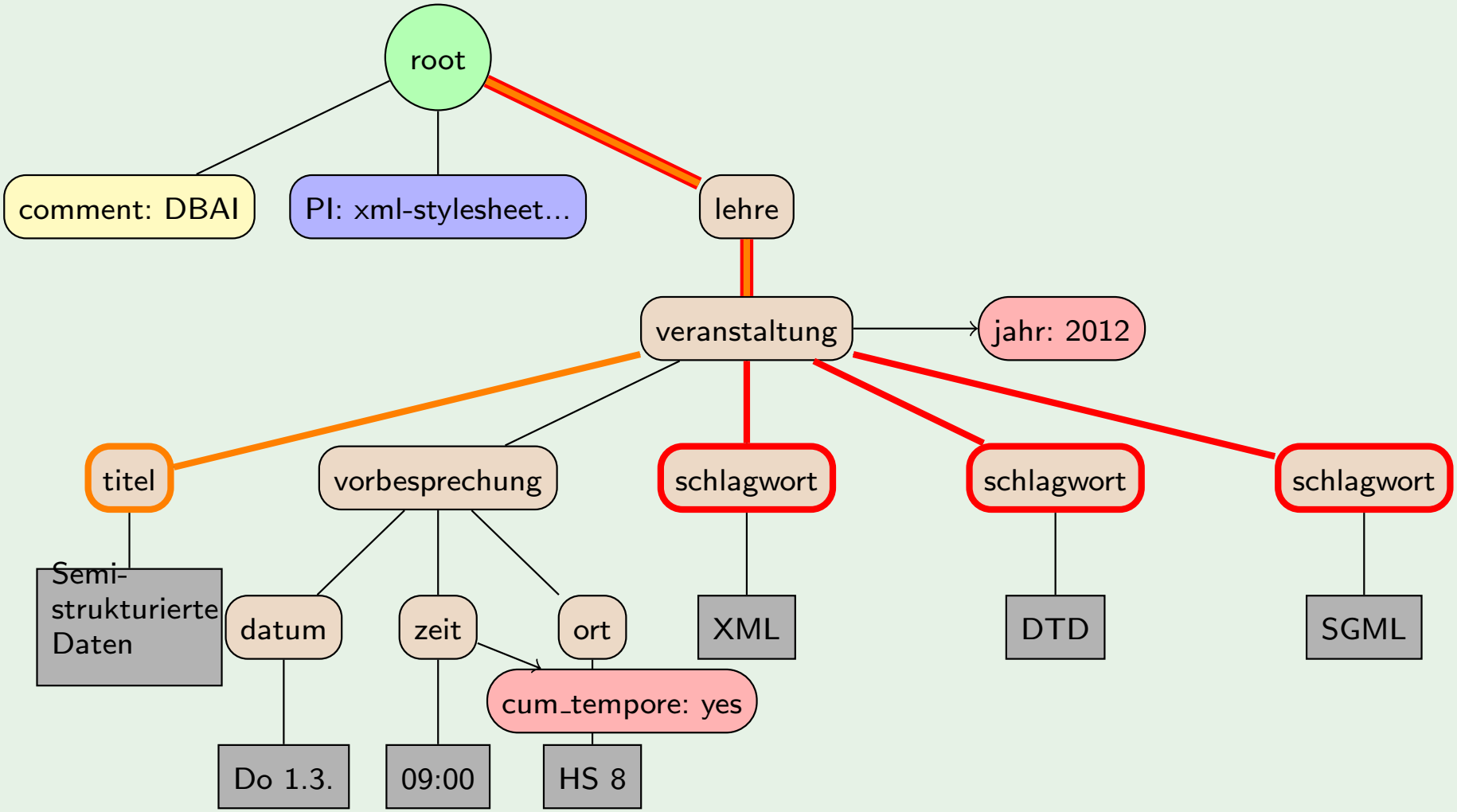
```
//zeit[@cum_tempore="yes"]
/descendant-or-self::node()/zeit[attribute::cum_tempore="yes"]

schlagwort[2]
child::schlagwort[position()=2]

schlagwort[last()]
child::schlagwort[position()=last()]
```

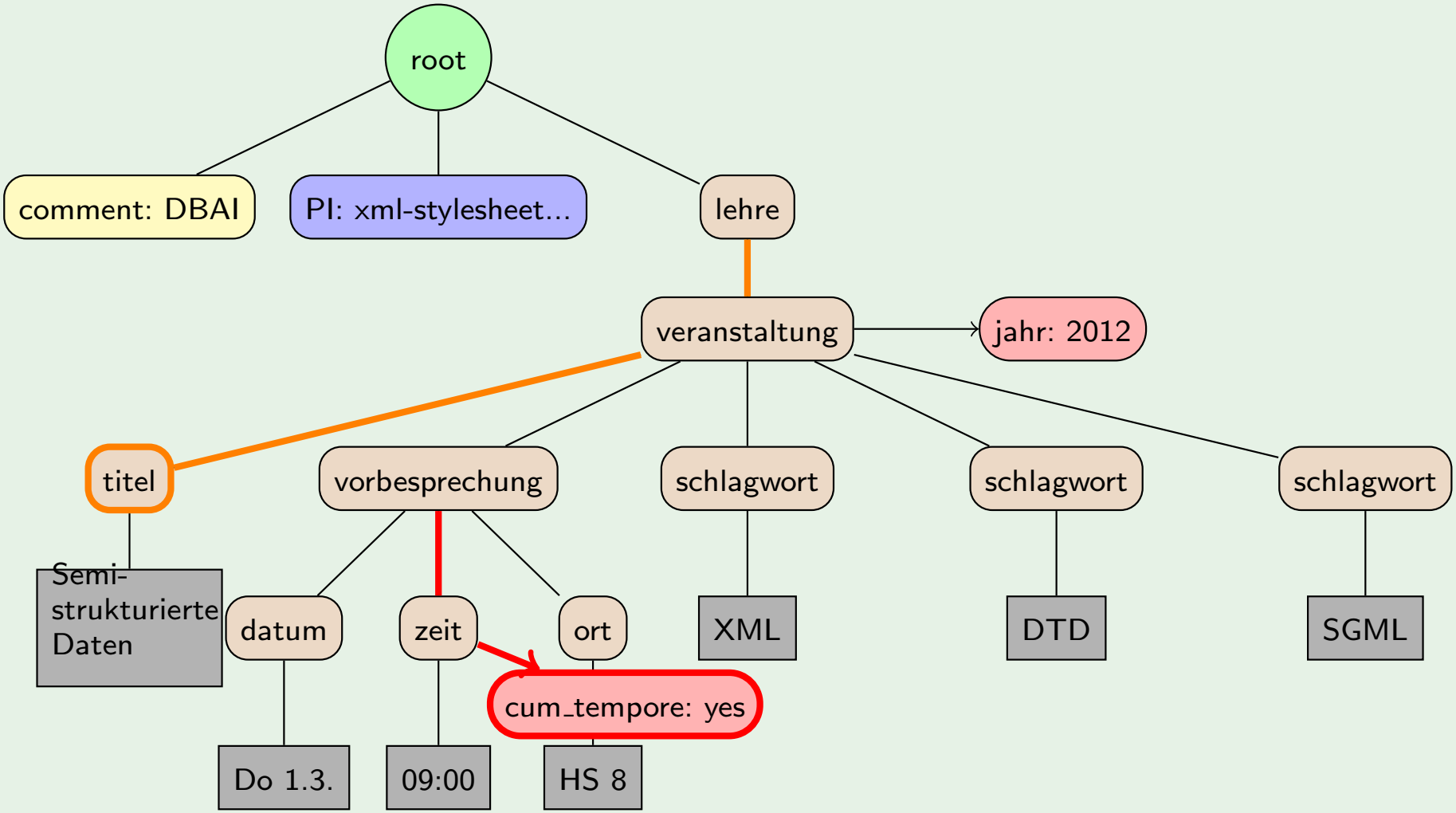
Beispiel (absolute Pfade)

```
/lehre/veranstaltung/titel
/lehre/veranstaltung/schlagwort
```



Beispiel (relative Pfade)

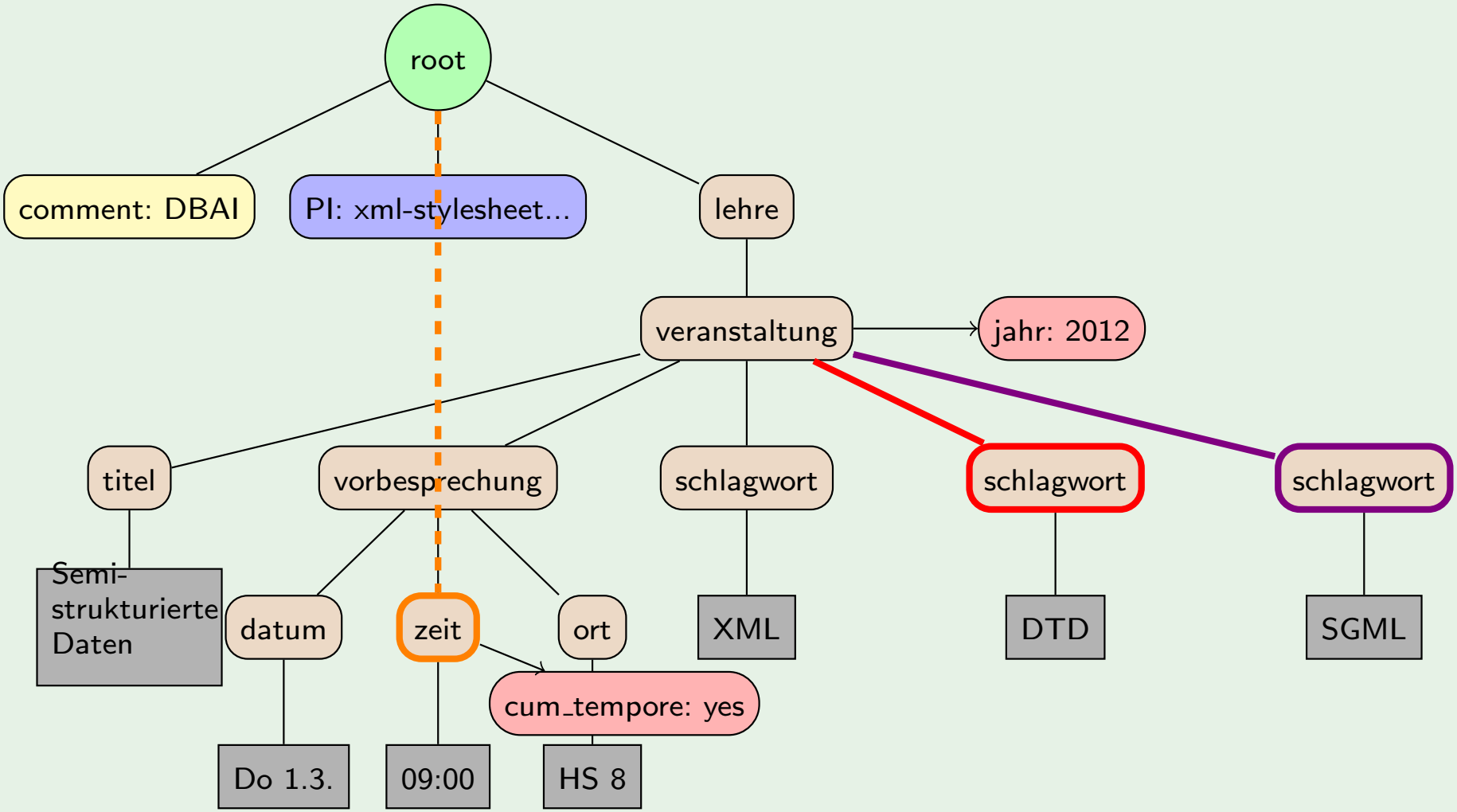
für context node "`lehre`": `veranstaltung/titel`
für context node "`vorbesprechung`": `zeit/@cum_tempore`



Beispiel (Verwendung von Filtern)

`//zeit[@cum_tempore="yes"]`

für context node "`veranstaltung`": `schlagwort[2]` `schlagwort[last()]`



Steps

- Ein Pfad setzt sich aus beliebig vielen Schritten zusammen (= steps)
- Bestandteile eines steps:

Achse	Richtung, in die navigiert wird
Node-Test	Typ bzw. Name der gewünschten Knoten
Prädikate	keine, eine oder mehrere Filterbedingungen

Beispiele

- `child::schlagwort[2]`
- `parent::*`
- `preceding-sibling::text()[last()]`
- `following::node()[@cum_tempore="yes"][1]`
- `attribute::jahr`
- `descendant::processing-instruction()`

Achsen(1)

`self`	der Knoten selbst
`child`	alle Kindknoten
`descendant`	alle Nachfahren
`descendant-or-self`	alle Nachfahren und der Knoten selbst
`parent`	Elternknoten
`ancestor`	alle Vorfahren
`ancestor-or-self`	alle Vorfahren und der Knoten selbst

Achsen(2)

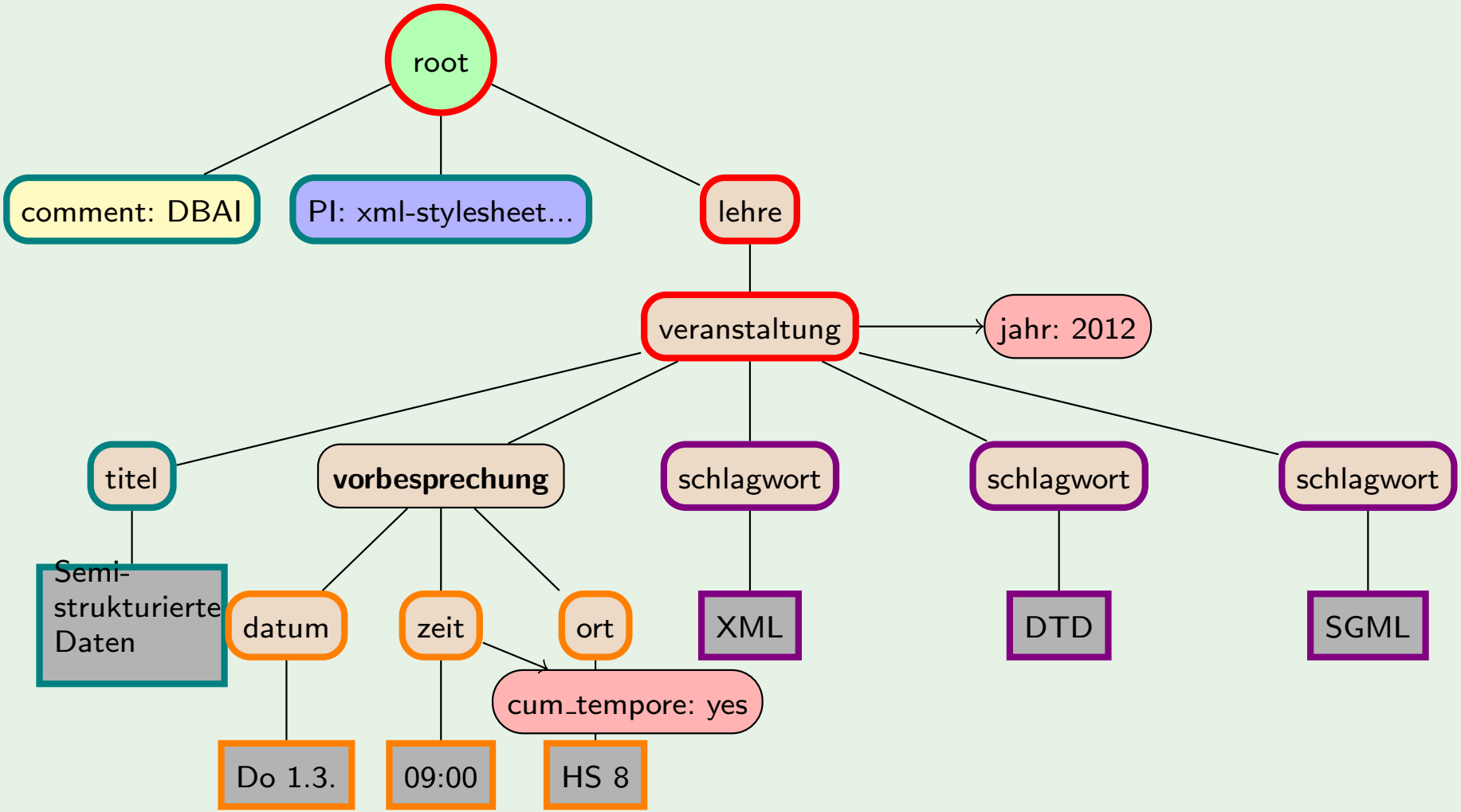
`following`	alle in document order nachfolgenden Knoten außer eigene Nachfahren
`following-sibling`	alle nachfolgenden Geschwisterknoten
`preceding`	alle in document order vorangegangenen Knoten außer eigene Vorfahren
`preceding-sibling`	alle vorangegangenen Geschwisterknoten
`attribute`	alle Attribute eines Elements
`namespace`	alle Namespaces eines Elements (deprecated in XPath 2.0, es können die Funktionen `fn:in-scope-prefixes` und `fn:namespace-uri-for-prefix` verwendet werden)

`ancestor(-or-self)` und `preceding(-sibling)` erzeugen Sequences in reverse document order

Beispiel (Dokumentenbaum: Achsen)

für context node "vorbesprechung":

ancestor descendant preceding following



Node Tests(1)

Üblicherweise: Angabe eines Namens (Name Test)

- Name und Knotentyp müssen einander entsprechen
  z.B. `/child::lehre/descendant::zeit`
- Wildcardtest mittels `*`
  z.B. `child::*` gibt alle Kind-Elementknoten zurück
  `attribute::*` gibt alle Attributknoten zurück
- Knoten mit bestimmtem Präfix
  z.B. `praefix:*` steht für beliebige qual. Namen mit Präfix `praefix`

Node Tests(2)

Überprüfen des Knotentyps (Kind Test)

- `comment()`: alle Kommentarknoten
  z.B. `/child::lehre/child::comment()`
- `text()`: alle Textnoten
- `attribute()`: alle Attributknoten
- `attribute(*, xs:decimal)`: alle Attributknoten mit Typ `xs:decimal`
- `node()`: alle Knoten
- ...

Abkürzungen

- weglassen der Achse entspricht der child-Achse
  z.B. `zeit` entspricht `child::zeit`
- `.` = Context Node (entspricht `self::node()`)
- `..` = parent des Context Node (entspricht `parent::node()`)
- `//` = Nachfahren und Context Node
  (entspricht `/descendant-or-self::node()/`)
- `@` = Abkürzung für Attributachse
  z.B. `@*` entspricht `attribute::*`

Beispiele

`./titel`
: alle `titel`-Elemente im momentanen Kontext
(äquivalent zu `titel` bzw. `child::titel`)

`/titel`
: selektiert `titel`, falls es das Dokumentelement ist

`.//titel`
: starte vom aktuellen context node und selektiere alle `titel`-Elemente, die tiefer liegen (also relativ): eigentlich
`descendant-or-self::node()/titel` **(vgl.** `descendant::titel`**)**

`//titel`
: starte von Wurzel und selektiere alle `titel`-Elemente, die tiefer liegen (also absolut)

`//zeit/../*`
: liefert alle `zeit`-Elemente plus deren Geschwister

`/lehre/*/schlagwort`
: ist äquivalent zu `/child::lehre/child::*/child::schlagwort`

Filter (Predicates)

- Angabe einer beliebigen XPath Expression in eckigen Klammern
  z.B. `//veranstaltung/schlagwort[position() >= 2]`
- die Filterbedingung `[x]` wird für jedes Element aus der Inputsequenz folgendermaßen ausgewertet:
  - `x` ist ein einzelner atomarer numerischer Wert:
    `[x]` entspricht `[position() = x]` (Vergleich mit der context position)
    z.B. `[3]` entspricht `[position() = 3]`
  - andernfalls wird die Funktion `fn:boolean(x)` aufgerufen:
    `[x]` entspricht `[boolean(x)]` (Auswertung des effektiven booleschen Werts)
    z.B. `[.//datum]` entspricht `[boolean(.//datum)]`

Beispiele

`veranstaltung[vorbesprechung]`
: selektiert alle `veranstaltung`-Elemente, die ein `vorbesprechung`-Element als Kind enthalten

`vorbesprechung[datum="Do 1.3."]`
: selektiert alle `vorbesprechung`-Elemente, die ein Subelement `datum` mit dem Textinhalt `Do 1.3.` haben

`zeit[@cum_tempore="yes"]`
: selektiert alle `zeit`-Elemente, die über das Attribut `cum_tempore` mit dem Wert `yes` verfügen

`vorbesprechung[not(zeit/@cum_tempore="yes")]`
: selektiert alle `vorbesprechung`-Elemente, die kein Subelement `zeit` mit dem Attribut `cum_tempore="yes"` haben

Beispiele

`vorbesprechung[datum and ort]`
: selektiert alle `vorbesprechung`-Elemente, die mindestens ein `datum` sowie einen ort als Kinder haben

`vorbesprechung[datum or ort]`
: selektiert alle `vorbesprechung`-Elemente, die mindestens ein `datum` oder einen ort als Kinder haben

`schlagwort[1]` bzw. `schlagwort[position()=1]`
: findet das erste `schlagwort`-Element

`schlagwort[last()]` bzw. `schlagwort[position()=last()]`
: findet das letzte `schlagwort`-Element

`schlagwort[2]/following::*`
: findet alle Elemente, die im XML-Dokument nach dem zweiten `schlagwort`-Element vorkommen (aber nicht als Nachfahre von `schlagwort`)

Filterlisten

- ein Step kann keine oder beliebig viele Predicates haben
- wenn `[b]` weder `position()` noch `last()` enthält, dann sind `[a and b]` und `[a][b]` identisch. Aber im Allgemeinen sind die beiden Ausdrücke verschieden, da nach der Auswertung eines Predicates der Context neu ermittelt wird.

Beispiele

`schlagwort[.="DTD"][2]`
wählt unter den Schlagwörtern mit Wert `DTD` das zweite aus
→ selektiert in diesem Fall die leere Knotenmenge

`schlagwort[2][.="DTD"]`
wählt das zweite Schlagwort aus, vorausgesetzt dass der Wert `DTD` ist
→ selektiert in diesem Fall das zweite Schlagwort

Auswertung von Steps

Auswertung eines Steps `achse::node-test[x][y]...`

- Zuerst wird die Sequence aufgrund von `achse::node-test` berechnet
  $\rightarrow$ Ergebnis $S = \{s_1, ..., s_k\}$
- Nun wird für jeden Kandidaten s_i das Predicate `x` bezüglich dem folgenden Context ausgewertet:
  - context-node $= s_i$
  - context-position $= i$
  - context-size $= k$
- Im Endergebnis liegen jene Knoten s_i, für die `x` den Wert `true` liefert.
- Diese Knoten werden dann bezüglich `[y]` ausgewertet, usw...

Funktionen und Operatoren

- Arithmetische Ausdrücke
- Vergleichsoperatoren
- Funktionen
- Kontext-Funktionen
- Funktionen auf Knoten
- Sequence-Funktionen
- String-Funktionen
- Funktionen auf Zahlen
- Boolesche Funktionen
- Konvertierungsfunktionen

Arithmetische Ausdrücke

- XPath unterstützt Arithmetische Ausdrücke in der gewohnten unären und binären Form
- Operatoren: `+`, `-`, `*`, `div`, `idiv` und `mod`
- damit - als Operator erkannt wird muss vor dem Operator ein Leerzeichen stehen
  z.B.: 'a-b' wird als Name, 'a - b' bzw. 'a -b' wird als arithmetischer Ausdruck interpretiert
- `div` ist die Dezimaldivision ('/' hat bereits eine andere Bedeutung)
- `idiv` ist die ganzzahlige Division

Beispiel

- `-3 div 2` liefert als Ergebnis `-1.5`
- `-3 idiv 2` liefert als Ergebnis `-1`

Vergleichsoperatoren

3 unterschiedliche Arten von Vergleichsoperatoren:

Wertevergleich	`eq`, `ne`, `lt`, `le`, `gt` und `ge`
allgemeiner Vergleich	`=`, `!=`, `<`, `<=`, `>` und `>=`
Knotenvergleich	`is`, `<<` und `>>`

Wertevergleich

- Operatoren: `eq`, `ne`, `lt`, `le`, `gt` und `ge`
- wird verwendet, um zwei einzelne Werte zu vergleichen
- bei Sequenzen mit mehr als einem Item wird ein Fehler geworfen

Beispiele

```
/lehre/veranstaltung/titel eq 'Semistrukturierte Daten'
```
liefert in unserem Beispiel `true`

```
/lehre/veranstaltung/schlagwort eq 'DTD'
```
liefert in unserem Beispiel einen Fehler, da die `schlagwort`-Sequenz mehrere Items enthält

```
/lehre/veranstaltung/@jahr lt '2000'
```
liefert `true`, sofern der Wert des Attributes `jahr` kleiner 2000 ist

Allgemeiner Vergleich

- Operatoren: =, !=, <, <=, > und >=
- Vergleich von Sequenzen mit beliebig vielen Items möglich
- Auswertung von Ausdrücken der Form `x genComp y`:
  - Anwendung des exists-Quantors
  - der Wert jedes Items aus `x` wird mit jedem Itemwert aus `y` verglichen (entsprechend dem Operator `genComp`)
  - liefert einer der Vergleiche `true`, so liefert der gesamte Ausdruck `true`

Beispiele

```
/lehre/veranstaltung/titel != 'Semistrukturierte Daten'
```
liefert in unserem Beispiel `false`

```
/lehre/veranstaltung/schlagwort = 'DTD'
```
liefert in unserem Beispiel `true`, da der Vergleich des Wertes der zweiten `schlagwort`-Items mit XML `true` liefert

```
//* = 'DTD'
```
liefert in unserem Beispiel `true`, da einer der Text Nodes im Dokumentenbaum den Wert `DTD` enthält

Knotenvergleich

- Operatoren: `is`, `<<` und `>>`
- dient zum Vergleich von zwei Knoten `x nodeComp y`
- `is` liefert `true`, wenn `x` der selbe Knoten wie `y` ist
- das Ergebnis von `<<` und `>>` wird von der document order bestimmt
  - `<<` liefert `true`, wenn `x` Vorgänger von `y` ist (anchestor oder preceding)
  - `>>` liefert `true`, wenn `x` Nachfolger von `y` ist (descendant oder following)

Beispiele

```
//veranstaltung is /lehre/veranstaltung
```
liefert in unserem Beispiel `true` (bei mehreren `veranstaltung`-Knoten im Baum, würde ein Fehler geliefert werden)

```
//schlagwort[1] << //schlagwort[2]
```
liefert in unserem Beispiel offensichtlich `true`

```
//schlagwort[4] >> //schlagwort[1]
```
liefert einen Fehler (da eine leere Sequenz mit einem Schlagwort verglichen wird)

Funktionen

- die beschriebenen Funktionen und Operatoren können in XPath 2.0, XQuery 1.0 und XSLT 2.0 verwendet werden
- XPath unterstützt nur Built-in Functions
- in XQuery können eigene user-defined functions geschrieben werden
- Funktionen haben folgende Funktionssignatur:

```
fn:function-name($parameter-name as parameter-type, ...)
    as return-type
```

`fn`	Default Präfix für Built-in Functions verweist auf "http://www.w3.org/2005/xpath-functions"
`function-name`	Name der Funktion
`$parameter`	Funktionen können eine beliebige Anzahl an Argumenten als Parameter erhalten
`return-type`	Rückgabetyp der Funktion

Kontext-Funktionen

- `fn:position() as xs:integer`
  Knotenposition in einer Knotenmenge (d.h.: context-position)
- `fn:last() as xs:integer`
  Gesamtzahl der zuletzt selektierten Knoten (d.h.: context-size)
- `fn:current-dateTime() as xs:dateTime`
  aktuelles Datum und Uhrzeit
- `fn:current-date() as xs:date`
  aktuelles Datum
- `fn:current-time() as xs:time`
  aktuelle Uhrzeit

Funktionen auf Knoten

- `fn:name($arg as node()?) as xs:string`
  Qualifizierter Name (d.h.: NS-Präfix + local-name) des Knoten `$arg` (falls kein Knoten angegeben: Name des aktuellen context-node)
- `fn:local-name($arg as node()?) as xs:string`
  Lokaler Name des Knoten `$arg` (falls kein Knoten angegeben: Name des aktuellen context-node)
- `fn:namespace-uri($arg as node()?) as xs:anyURI`
  Namespace URI des Knoten `$arg` (falls kein Knoten angegeben: Namespace URI des aktuellen context-node)
- `fn:lang($testlang as xs:string?, $node as node())`
  `as xs:boolean`
  liefert `true`, wenn die Sprache des context-node oder `$node` (laut xml:lang-Attribut) dieselbe Sprache oder eine Subsprache des Inputstring `$testlang` ist

Sequence-Funktionen(1)

- `fn:index-of($seqParam as xs:anyAtomicType*, $srchParam as xs:anyAtomicType) as xs:integer*`
  Positionen von $srchParam in $seqParam als Sequenz von Integern
- `fn:empty($arg as item()*) as xs:boolean`
  überprüft, ob $arg eine leere Sequenz ist
- `fn:exists($arg as item()*) as xs:boolean`
  Gegenstück zu `fn:empty`
- `fn:distinct-values($arg as xs:anyAtomicType*) as xs:anyAtomicType*`
  eliminiert alle Duplikate in $arg entsprechend des eq-Operators

Sequence-Funktionen(2)

- `fn:insert-before($target as item()*, $position as xs:integer, $inserts as item()*) as item()*`
  erzeugt eine Sequenz mit `$inserts` an `$position` in `$target` eingefügt
- `fn:remove($target as item()*, $position as xs:integer) as item()*`
  entfernt das Item an der Position `$position`
- `fn:reverse($arg as item()*) as item()*`
  die Reihenfolge der Items wird umgedreht
- `fn:subsequence($sourceSeq as item()*, $startingLoc as xs:double, $length as xs:double) as item()*`
  erzeugt eine Sub-Sequenz aus `$sourceSeq`, beginnend bei `$startingLoc` (`$length` ist optional)

Sequence-Funktionen(Aggregatfunktionen)

- `fn:count($arg as item()*) as xs:integer`
  Anzahl der Knoten
- `fn:avg($arg as xs:anyAtomicType*) as xs:anyAtomicType?`
  Durchschnitt der Werte (`fn:sum div fn:count`)
- `fn:max($arg as xs:anyAtomicType*) as xs:anyAtomicType?`
  jenes Item mit dem größten Wert
- `fn:min($arg as xs:anyAtomicType*) as xs:anyAtomicType?`
  jenes Item mit dem kleinsten Wert
- `fn:sum($arg as xs:anyAtomicType*) as xs:anyAtomicType?`
  Summe der Werte aus `$arg`

String-Funktionen(1)

- `fn:concat($arg1 as xs:anyAtomicType?, $arg2 as xs:anyAtomicType?, ...) as xs:string`
  Konkatenation von zwei oder mehr Strings
- `fn:starts-with($arg1 as xs:string?, $arg2 as xs:string?) as xs:boolean`
  liefert `true`, wenn das erste Argument mit dem zweiten beginnt
- `fn:contains($arg1 as xs:string?, $arg2 as xs:string?) as xs:boolean`
  liefert `true`, wenn das erste Argument das zweite enthält
- `fn:substring($sourceString as xs:string?, $startingLoc as xs:double, $length as xs:double) as xs:string`
  liefert Substring von `$sourceString` beginnend bei `$startingLoc` (`$length` ist optional)
  z.B. `fn:substring('abcdef', 2, 3)` liefert 'bcd'

String-Funktionen(2)

- `fn:substring-before($arg1 as xs:string?, $arg2 as xs:string?) as xs:string`
  liefert Substring von $arg1 vor dem ersten Auftreten von $arg2 (sonst den leeren String)
- `fn:string-length($arg as xs:string?) as xs:integer`
  liefert die Stringlänge von $arg (bzw. den Stringwert des context node, falls kein Argument angegeben)
- `fn:normalize-space($arg as xs:string?) as xs:string`
  normalisiert $arg (bzw. den Stringwert des context node) bezüglich Whitespaces
- `fn:translate($arg as xs:string?, $mapString as xs:string, $transString as xs:string) as xs:string`
  Zeichenweise Transformation von $arg
  z.B. `fn:translate('---aaa---', 'abc', 'ABC')` liefert '---AAA---'

Funktionen auf Zahlen

- `fn:abs($arg as numeric?) as numeric?`
  liefert den absoluten Wert von `$arg`
- `fn:ceiling($arg as numeric?) as numeric?`
  rundet nach oben
- `fn:floor($arg as numeric?) as numeric?`
  rundet nach unten
- `fn:round($arg as numeric?) as numeric?`
  rundet zur nächstgelegenen ganzen Zahl
- `fn:ceiling($arg as numeric?, $precision as xs:integer) as numeric?`
  rundet zur nächstgelegenen Zahl mit `$precision` Nachkommastellen (bei negativer `$precision` wird vor der Nachkommastelle gerundet)

Boolesche Funktionen

- `fn:true() as xs:boolean`
  liefert den Wert `true` (äquivalent zu `xs:boolean('1')`)
- `fn:false() as xs:boolean`
  liefert den Wert `false` (äquivalent zu `xs:boolean('0')`)
- `fn:not($arg as item()*) as xs:boolean`
  logische Negation

Konvertierungsfunktionen

- `fn:number($arg as xs:anyAtomicType?) as xs:double`
  konvertiert `$arg` (bzw. den aktuellen context-node) in eine Zahl, falls Konvertierung nicht möglich → `NaN`
- `fn:string($arg as item()*) as xs:string`
  Typkonvertierung in einen String (siehe Kapitel String-Wert eines Knotens)
- `fn:boolean($arg as item()*) as xs:boolean`
  Typkonvertierung in einen booleschen Wert nach folgenden Regeln:
  - `$arg` ist eine leere Sequenz → `false`
  - erstes Item der Sequenz ist ein Knoten → `true`
  - xs:boolean → Wert des Boolean
  - leerer String → `false`
  - Zahl → `false`, falls `NaN` oder 0, sonst `true`

Zusammenfassung und Links

- XPath dient zum Navigieren in XML-Dateien
- Grundlage für XQuery, XSLT und XPointer
- XPath 2.0 Recommendation
  - siehe http://www.w3.org/TR/xpath20/
- basiert auf dem XQuery 1.0 und XPath 2.0 Data Model (XDM)
  - siehe http://www.w3.org/TR/xpath-datamodel/
- wichtigste Expression: Paths
- unterstützt eine Vielzahl von Built-in Functions
  - siehe http://www.w3.org/TR/xpath-functions/
- Tools:
  - XPathWay (Aufruf mittels `java -jar xpathway.jar`):
    - http://www.dbai.tuwien.ac.at/education/ssd/current/xpathway/xpathway.zip
  - SAXON:
    - http://saxon.sourceforge.net/