

# XML Datenstruktur mit XPath Achsen

Sven Eric Panitz

21. April 2019

## Inhaltsverzeichnis

<b>1</b>	<b>XML</b>	<b>1</b>
<b>2</b>	<b>XML Datenstruktur</b>	<b>2</b>
2.1	Datenstruktur . . . . .	2
2.2	Instanz von Show . . . . .	3
2.3	Instanz von EQ . . . . .	3
2.4	Konstruktorfunktionen . . . . .	3
<b>3</b>	<b>XPath</b>	<b>4</b>
<b>4</b>	<b>Aufgaben</b>	<b>5</b>
<b>5</b>	<b>Lernzuwachs</b>	<b>8</b>

## 1 XML

XML ist ein vom W3C Konsortium als Empfehlung herausgegebener Standard für eine hierarchische Datenstruktur[BPSM<sup>+</sup>08].

XML haben eine Baumstruktur und kennen primär innerhalb des Baumes zwei Arten von Knoten.

- Textknoten. Diese sind immer Blätter des Baumes. Sie können keine Kinder haben. Sie enthalten einen Text.
- Elementknoten. Die Wurzel muss ein Elementknoten sein. Ein Elementknoten hat einen Tagnamen. Er kann Kinderknoten enthalten, die auch gemischt aus Textknoten und Elementknoten bestehen können.

Zusätzlich enthalten Elementknoten in Form von Attributen einen Map, also eine Liste von Schlüssel-Wertpaaren.

Die W3C Empfehlung enthält die Spezifikation der syntaktischen Serialisierung der Baumstruktur eines XML Dokuments.

Wir begnügen uns in dieser Aufgabe mit einem kleinen Beispieldokument.

#### XML Beispiel

```
<?xml version="1.1"?>
<exercise lang="LiterateHaskell" extension="lhs">
  <name>XML Datenstruktur mit XPath Achsen</name>
  <text>
    <div>
      <p>Studieren Sie das Aufgaben-Papier zu XML und lösen Sie die
        dort enthaltenen Aufgaben.</p>
    </div>
  </text>
</exercise>
```

## 2 XML Datenstruktur

In diesem Übungsblatt sei eine kleine Datenstruktur für XML-Dokumente gegeben.

#### Haskell: XML

```
> module XML where
```

### 2.1 Datenstruktur

Wir sehen zwei Arten von XML-Knoten vor. Textknoten und Elementknoten. Ein Knoten hat als ersten Parameter einen Verweis auf den Elternknoten, sofern es sich selbst nicht um die Wurzel handelt. Der zweite Parameter beider Konstruktoren zeigt an, um das wie viele Kind es sich handelt, wenn der Knoten einen Elternknoten hat. Wurzelknoten sind dabei mit  $-1$  markiert.

#### Haskell: XML

```
> data XML
> = Text (Maybe XML) Index String
> |Element (Maybe XML) Index Name Attributes ChildNodes
```

Ein Textknoten hat nur den eigentlichen Text. Ein Elementknoten einen Tagnamen, eine Attributliste und die Liste der Kinder.

Dabei wurden folgende Typsynonyme verwendet

### Haskell: XML

```
> type Name = String
> type ChildNodes = [XML]
> type Attribute = (Key,Value)
> type Attributes = [Attribute]
> type Key = String
> type Value = String
> type Index = Int
```

## 2.2 Instanz von Show

Da mit den Verweis ein den Elternknoten in einem Knoten, die XML Datenstruktur zyklisch ist, können nicht die per `deriving` generierten Versionen von `Show` und `Eq` verwendet werden und es sind somit eigene manuell Instanzen zu definieren.

Eine simple Instanz der Typklasse `Show` serialisiert die XML-Knoten in XML-Syntax.

### Haskell: XML

```
> instance (Show XML) where
>   show (Text _ _ txt) = txt
>   show (Element _ _ name attrs cs)
>     = "<" ++ name ++ concatMap showAttr attrs ++ ">"
>       ++ concatMap show cs ++ "</" ++ name ++ ">"
>   where
>     showAttr (k,v) = " " ++ k ++ "=" ++ "\"" ++ v ++ "\""
```

## 2.3 Instanz von EQ

Eine eigene Instanz für `Eq` prüft auf Gleichheit. Der Einfachheit halber, wird für die Gleichheit von Elementknoten verlangt, dass sie die gleiche Attributliste haben. Das ist semantische für Attribute in XML-Knoten nicht korrekt, da Attributknoten keine feste Reihenfolge haben.

### Haskell: XML

```
> instance (Eq XML) where
>   (Text _ i1 txt1) == (Text _ i2 txt2) = txt1 == txt2
>   (Element _ i1 name1 attrs1 cs1) == (Element _ i2 name2 attrs2 cs2)
>     = name1 == name2 && attrs1 == attrs2 && cs1 == cs2
>   _ == y = False
```

## 2.4 Konstruktorfunktionen

Eine Konstruktorfunktion zur Erzeugung von Elementen sei gegeben. Diese setzt den Verweis auf den Elternknoten korrekt und die Indexangabe über die Position in der Kinderliste. Der komplette Kindbaum muss dabei neu erzeugt werden.

Haskell: XML

```
> mkElement :: Name -> Attributes -> [XML] -> XML
> mkElement n as cs = r
>   where
>     r = Element Nothing (-1) n as (map (setParent r)(zip [0..] cs))
```

XML-Dokumente sind nicht nur von Elternknoten zu Kinderknoten verlinkt, sondern auch von Kinderknoten zurück zu Elternknoten. Wenn ein Element erzeugt wird, so sind den Kinderknoten dieses Elements die Referenzen auf den Elternknoten zu setzen.

Haskell: XML

```
>   setParent p (n, (Text _ _ txt) ) = Text (Just p) n txt
>   setParent p (n, (Element _ _ na as cs)) = r
>   where
>     r = Element (Just p) n na as (map (setParent r)(zip [0..] cs))
```

Praktischer Weise sei auch eine Konstruktorfunktion für Textknoten vorgesehen.

Haskell: XML

```
> mkText :: String -> XML
> mkText = Text Nothing (-1)
```

## 3 XPath

Zur Navigation und zur Selektion von Knoten innerhalb eines XML-Dokuments gibt es die Sprache XPath. Auch diese wurde vom W3C als Empfehlung definiert[CD99].

Der zentrale Begriff innerhalb der Sprache XPath sind die XPath-Achsen. Diese selektieren ausgehend von einem Knoten innerhalb eines XML-Dokuments eine Liste von Knoten. Die Achsen schauen meist in eine Richtung.

So schauen die Achsen *child*, *descendant* und *descendant-or-self* ausgehend von einem Knoten in Richtung der Blätter.

*parent*, *ancestor* und *ancestor-or-self* schauen ausgehend von einem Knoten in Richtung der Wurzel.

*following-sibling* schaut auf einer Ebene nach rechts.

*preceding-sibbling* schaut auf einer Ebene nach links.

Die beiden Achsen *following* und *preceding* betrachten das Dokument in seiner serialisierten Form und selektieren alle Knoten die nach dem Ausgangsknoten beginnen bzw. vor dem Ausgangsknoten enden.

## 4 Aufgaben

Implementieren Sie Funktionen für die XPath Achsen. Eine XPath-Achse selektiert innerhalb eines Elements ausgehend von einem Knoten eine Liste von Knoten des XML-Baums. Dieses lässt sich als Typsynonym ausdrücken.

Haskell: XML

```
> type XPathAchs = XML -> [XML]
```

### Aufgabe 1

Implementieren Sie die Achse *self*:

Haskell: XML

```
> self :: XPathAchs
> self n = []
```

### Aufgabe 2

Implementieren Sie die Achse *parent*:

Haskell: XML

```
> parent :: XPathAchs
> parent _ = []
```

### Aufgabe 3

Implementieren Sie die Achse *ancestor*:

Haskell: XML

```
> ancestor :: XPathAchs
> ancestor x = []
```

#### Aufgabe 4

Implementieren Sie die Achse *ancestor-or-self*:

Haskell: XML

```
> ancestorOrSelf :: XPathAchsIs  
> ancestorOrSelf _ = []
```

#### Aufgabe 5

Implementieren Sie die Achse *child*:

Haskell: XML

```
> child :: XPathAchsIs  
> child _ = []
```

#### Aufgabe 6

Implementieren Sie die Achse *descendant*:

Haskell: XML

```
> descendant :: XPathAchsIs  
> descendant x = []
```

#### Aufgabe 7

Implementieren Sie die Achse *descendant-or-self*:

Haskell: XML

```
> descendantOrSelf :: XPathAchsIs  
> descendantOrSelf x = []
```

### Aufgabe 8

Implementieren Sie die Achse *following-sibling*:

Haskell: XML

```
> followingSibling :: XPathAchsIs  
> followingSibling _ = []
```

### Aufgabe 9

Implementieren Sie die Achse *preceding-sibling*:

Haskell: XML

```
> precedingSibling :: XPathAchsIs  
> precedingSibling _ = []
```

### Aufgabe 10

Implementieren Sie die Achse *Following*:

Haskell: XML

```
> following :: XPathAchsIs  
> following xml = [ ]
```

### Aufgabe 11

Implementieren Sie die Achse *preceding*:

Haskell: XML

```
> preceding :: XPathAchsIs  
> preceding xml = [ ]
```

### Aufgabe 12

Für die XML-Verarbeitung gibt es in Haskell das Paket *HaXml* [Wal18]. Machen Sie sich mit dem Paket vertraut. Schauen Sie sich die Implementierung an und vergleichen Sie diese mit der Implementierung unserer Aufgabe.

### Projektidee 1

Mit den Achsen ist der erste Schritt für einen XPath-Prozessor implementiert. Wie wäre es einen kompletten XPath-Prozessor und schließlich eine XSLT Transformation in Haskell zu implementieren.

## 5 Lernzuwachs

Folgende Erkenntnisse sollten sich nach Studium dieses Kurses gesammelt haben.

- Man kann in Haskell auch mehrfach verkettete Datenstrukturen definieren, braucht hierfür aber entsprechende Konstruktorfunktionen.
- Für mehrfach verkettete also zyklische Strukturen können nicht die automatisch generierten Version der Gleichheit oder von `show` verwendet werden.
- Die Haskellimplementierung der XPath-Achsen ist wahrscheinlich präziser und stellt eine eindeutigere Spezifikation dar, als die eigentliche Spezifikation in der W3C Empfehlung.

## Literatur

- [BPSM<sup>+</sup>08] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, “Extensible markup language (xml) 1.0 (fifth edition),” World Wide Web Consortium, Recommendation REC-xml-20081126, November 2008. [Online]. Available: <http://www.w3.org/TR/2008/REC-xml-20081126>
- [CD99] J. Clark and S. DeRose, *XML path language (XPath) version 1.0. W3C Recommendation 16 November 1999*, internet, W3C Std., 1999. [Online]. Available: <http://www.w3.org/TR/xpath/>
- [Wal18] Wallace, Malcolm, “Haxml: Utilities for manipulating xml documents,” 2018, [Online; accessed 20-April-2019]. [Online]. Available: <http://hackage.haskell.org/package/HaXml>