Ein Parser für XML-Dokumente in Haskell

Sven Eric Panitz

21. April 2019

Inhaltsverzeichnis

1 XML Grammatik **1**

2 XML Parser **2**

3 Lernzuwachs **5**

1 XML Grammatik

In diesem Übungsblatt soll ein Parser für XML-Dokumente geschrieben werden. Um den Umfang einer Übungsaufgabe nicht zu sprengen, konzentrieren wir uns auf eine Untermenge der kompletten Syntax. Wir werden Elemente mit Attributen und Kindern aus Text und Elementknoten parsen. Hingegem werden wir *character entities*, *CData section*, Kommentare, *processing instructions* und die Definition einer DTD weglassen.

Die Grammatik für XML ist durch das W3C in einer Empfehlung definiert[BPSM+08]. Die Definition erfolgt dort in einer erweiterten Backus-Naur-Form. In den Regeln steht das Fragezeichen für eine Option, das Multiplikationszeichen für eine 0 bis n-fache Wiederholung und das Additionszeichen für eine 1 bis n-fache Wiederholung.

Die Grammatik ist auch Unicode-Zeichen definiert. Es gibt also keine eigene lexikographische Beschreibung.

Bemerkenswert ist, dass weißer Zwischenraum in der Grammatik explizit definiert ist. Es gibt ein Nichtterminalzeichen <*S*>, das für Zwischenraum steht.

Es folgen die relevanten Regeln der XML-Grammatik, für die Aufgabe.

⟨*NameStartChar*⟩ ::= ':' | [A-Z] | '_' | [a-z] | [#xC0-#xD6] | [#xD8-#xF6] | [#xF8-#x2FF] | [#x370-#x37D] | [#x37F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F]

| [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]

⟨*NameChar*⟩ ::= ⟨*NameStartChar*⟩ | '-' | '.' | [0-9] | #xB7 | [#x0300-#x036F] | [#x203F-#x2040]

⟨*Name*⟩ ::= ⟨*NameStartChar*⟩ (⟨*NameChar*⟩)*

⟨*S*⟩ ::= (#x20 | #x9 | #xD | #xA)+

⟨*ETag*⟩ ::= '</' ⟨*Name*⟩ ⟨*S*⟩? '>'

⟨*Reference*⟩ ::= ⟨*EntityRef*⟩ | ⟨*CharRef*⟩

⟨*EntityRef*⟩ ::= '&' ⟨*Name*⟩ ';'

⟨*PEReference*⟩ ::= '%' ⟨*Name*⟩ ';'

⟨*CharRef*⟩ ::= '&#' [0-9]+ ';' | '&#x' [0-9a-fA-F]+ ';'

⟨*Attribute*⟩ ::= ⟨*Name*⟩ ⟨*Eq*⟩ ⟨*AttValue*⟩

⟨*Eq*⟩ ::= ⟨*S*⟩? '=' ⟨*S*⟩?

⟨*STag*⟩ ::= '<' ⟨*Name*⟩ (⟨*S*⟩ ⟨*Attribute*⟩)* ⟨*S*⟩? '>'

⟨*element*⟩ ::= ⟨*EmptyElemTag*⟩ | ⟨*STag*⟩ ⟨*content*⟩ ⟨*ETag*⟩

⟨*content*⟩ ::= ⟨*CharData*⟩? ((⟨*element*⟩ | ⟨*Reference*⟩ | ⟨*CDSect*⟩ | ⟨*PI*⟩ | ⟨*Comment*⟩) ⟨*CharData*⟩?)*

⟨*CharData*⟩ ::= [^<&]* - ([^<&]* ']]>' [^<&]*)

⟨*AttValue*⟩ ::= '"' ([^<&"] | ⟨*Reference*⟩)* '"' | "'" ([^<&'] | ⟨*Reference*⟩)* "'"

2 XML Parser

Sie sollen jetzt einen Parser für eine Untermenge von XML schreiben. Hierzu verwenden wir die kleiner Parserbibliothek aus einer früheren Aufgabe und die Datenstruktur für XML, die wir in einer früheren Aufgabe entwickelt haben.

Haskell: ParseXML

```
> module ParseXML where
> import XML
> import OurParserLib
> import Data.Char
> import Data.List
```

nun dürfen Sie schrittweise Regeln der Grammatik umsetzen.

Aufgabe 1

Implementieren Sie den Parser, der Tag- und Attributnamen parst.
Ein solcher Name ist ein nichtleerer String, der mit einem Buchstaben (isAlpha) oder einen der Zeichen `'_'` und `':'` begint und gefolgt wird von einer Zeichenkette aus beliebigen Zeichen die kein Whitespace sind und nicht eines der Zeichen `'<', '/', '>', '\'', '='` und `'\"'`.

Haskell: ParseXML

```
> name :: Parser Char Name
> name xs = []
```

Aufgabe 2

Implementieren Sie einen Parser, der einen schließenden XML-Tag parst.
Das Ergebnisse sei der Tagname.
Beachten Sie dass nach `"</"` kein Whitespace stehen darf, aber nach dem Tagnamen beliebiger Whitespace stehen darf.

Haskell: ParseXML

```
> closeTag :: Parser Char Name
> closeTag xs = []
```

Aufgabe 3

Implementieren Sie einen Parser, der Attributwerte erkennt.
Ein Attributwert sei dabei eine entweder in doppelten Anführungszeichen " oder in einfachen Anführungszeichen eingeschlossene Zeichenkette, in der die verwendeten Anführungszeichen nicht verwendet werden dürfen.
Wir vernachlässigen Character-Entities.

Haskell: ParseXML

```
> value :: Parser Char Value
> value xs =  []
```

Aufgabe 4

Implementieren Sie einen Parser der Attribute erkennt.
Ein Attribute hatt einen Namen gefolgt vom Gleichheitszeichen, gefolgt von einem Attributwert. Vor und nach dem Gleichheitszeichen dürfen beliebig viele Weißzeichen stehen.

Haskell: ParseXML

```
> attribute :: Parser Char Attribute
> attribute xs = []
```

Aufgabe 5

Implementieren Sie einen Parser, der eine Folge von Attributen erkennt. Vor den Attributen dürfen beliebig viele Weißzeichen stehen.
Schließen Sie Attributlisten, in denen ein Schlüssel mehrfach vor kommt, aus.

Haskell: ParseXML

```
> attributes :: Parser Char [Attribute]
> attributes xs = []
```

Aufgabe 6

Implementieren Sie einen Parser, der ein öffnendes Element-Tag erkennt.
Ein Element-Tag beginnt mit dem Symbol `<` gefolgt von einen Namen. Anschließend kommt eine Attributeliste. Zum Ende kommt das Symbol `>`. Vor dem letzten Symbol darf belliebig viel Weißraum stehen.
Wir vernachlässigen direkt wieder schließende Elemente der Art `<tag/>`.

Haskell: ParseXML

```
> openTag :: Parser Char (String, [Attribute])
> openTag xs =  []
```

Aufgabe 7

Schreiben Sie einen Parser der beliebigen Text eines Textknotens erkennt.
Text sei eine Folge vin beliebigen Zeichen außer <.
Wir vernachlässigen also Character-Entities, CData-Sections und vieles mehr.

Haskell: ParseXML

```
> text :: Parser Char XML
> text xs = []
```

Aufgabe 8

Schreiben Sie einen Parser, der ein XML-Element erkennt.
Ein XML-Element hat ein öffnendes und ein schließendes Tag. Beide haben denselben Tagnamen, was Sie mit der Funktion `filtere` ausdrücken können.
Zwischen öffnenden und schließenden Tag steht eine Folge von beliebigen XMl-Knoten. Das können weitere Elemente oder Textknoten sein.

Haskell: ParseXML

```
> element :: Parser Char XML
> element xs = []

> node :: Parser Char XML
> node xs = []

> content :: Parser Char [XML]
> content xs = []
```

Projektidee 1

Soweit die Aufgabe. Nun juckt es doch eigentlich in den Fingern, die komplette XML-Grammatik zu parsen.

3 Lernzuwachs

Folgende Erkenntnisse sollte sich nach Studium dieses Kurses gesammelt haben.

- Mit einer sehr einfachen Kombinatorparserbibliothek lassen sich komplexe Grammatiken umsetzen.

- Die XML-Grammatik drückt nicht aus, dass das schließende Tag denselben Namen haben muss wie das öffnende Tag. Dieses ist im Anschluss nach dem Parsen zu überprüfen.

Literatur

[BPSM+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