Aufgabe 1: XML Familie [35]

Gegeben ist folgendes XML Schema. (Hinweis: die Zahlen am Beginn jeder Zeile sind Zeilennummern, auf die Sie sich bei der Lösung der Aufgabe beziehen können.)

```
 1 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 2
 3  <xsd:complexType name="autos_t">
 4   <xsd:sequence minOccurs="2" maxOccurs="unbounded">
 5    <xsd:element name="auto" type="auto_t" minOccurs="0"/>
 6    <xsd:element name="haendler" type="xsd:string" minOccurs="0"/>
 7   </xsd:sequence>
 8  </xsd:complexType>
 9
10  <xsd:complexType name="auto_t">
11   <xsd:sequence>
12    <xsd:element name="besitzer" type="besitzer_t"/>
13    <xsd:element name="modell" type="xsd:string"/>
14    <xsd:element name="bauart" type="bauart_t"/>
15   </xsd:sequence>
16   <xsd:attribute name="marke" type="xsd:string" use="required"/>
17   <xsd:attribute name="baujahr" type="xsd:integer" use="required"/>
18   <xsd:attribute name="info" type="xsd:string" use="optional"/>
19  </xsd:complexType>
20
21  <xsd:simpleType name="besitzer_t">
22   <xsd:restriction base="xsd:string">
23    <xsd:maxLength value="10"/>
24   </xsd:restriction>
25  </xsd:simpleType>
26
27  <xsd:simpleType name="bauart_t">
28   <xsd:restriction base="xsd:string">
29    <xsd:enumeration value="SUV"/>
30    <xsd:enumeration value="Kombi"/>
31    <xsd:enumeration value="Stufenheck"/>
32    <xsd:enumeration value="Cabrio"/>
33   </xsd:restriction>
34  </xsd:simpleType>
35
36  <xsd:element name="autos" type="autos_t">
37   <xsd:unique name="x">
38    <xsd:selector xpath="auto"/><xsd:field xpath="besitzer"/>
39   </xsd:unique>
40  </xsd:element>
41
42 </xsd:schema>
```

1a. Markieren Sie jene Stellen im folgenden XML Dokument, die in Bezug auf das gegebene XML Schema *ungültig* sind. Begründen Sie jeweils kurz, warum die Stelle ungültig ist. [13]

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <autos>
4    <auto marke="Ford" baujahr="2000">
5      <besitzer>M. Derntl</besitzer>
6      <modell>Focus Traveller 1.8i</modell>
7      <bauart>SUV Kombi</bauart>
8    </auto>
9
10   <auto marke="Hummer" jahr="2005">
11     <besitzer>A. Schwarzenegger</besitzer>
12     <bauart>SUV</bauart>
13     <modell/>
14   </auto>
15
16   <auto marke="Peugeot" baujahr="1996">
17     <besitzer>M. Derntl</besitzer>
18     <modell>306</modell>
19     <bauart>Stufenheck</bauart>
20   </auto>
21
22   <auto marke="Ferrari" baujahr="2008"/>
23
24   <auto marke="Aston Martin" info="Filmauto">
25     <besitzer>James Bond</besitzer>
26     <modell>DB 9</modell>
27     <bauart>Cabrio</bauart>
28     <bauart>SUV</bauart>
29     <radstand>3.5m</radstand>
30   </auto>
31
32 </autos>
```

1b. Erklären Sie, welche in dem XML Schema definierten Einschränkungen Sie mit einer DTD nicht definieren könnten (mit kurzer Begründung!) [8]

1c. Kreuzen Sie die *falschen* Aussagen an: [4]

- **O** Mit Attributgruppen kann man in XML Schema die Reihenfolge der Attribute eines XML Elements definieren.
- **O** Wird für ein Attribut kein Namespace Präfix angegeben, so befindet sich das Attribut im Default Namespace.
- **O** In einer WSDL Datei muss man immer ein XML Schema im <types> Abschnitt angeben.
- **O** In XSLT werden die im Stylesheet definierten Templates der Reihe nach auf das Quelldokument angewendet.
- **O** Der XPath Ausdruck `/*` kann mehrere Elemente als Ergebnis liefern.
- **O** Bei XML Elementnamen ist die Groß-/Kleinschreibung irrelevant.
- **O** Wohlgeformte XML Dokumente können niemals ungültigen Inhalt haben.

1d. Nehmen Sie an, es existiert ein XML Dokument, das gültig für das gegebene XML Schema ist. Erstellen Sie nun je einen XPath Ausdruck für folgende Abfragen:

i. Die Besitzer aller Autos der Marke “BMW”. [2]

ii. Die Baujahre aller Autos, deren Besitzer “Ali G.” heißt. [2]

iii. Das Modell des letzen in der Datei gespeicherten Autos. [1]

iv. Alle Baujahre in denen mindestens zwei Autos gebaut wurden. (Hinweis: dafür brauchen Sie XPath Achsen!) [3]

v. Die Anzahl der Autos der Bauart “Cabrio”, die vor dem Jahr 2000 gebaut wurden, und die nicht von der Marke “Mercedes” sind. [2]

Aufgabe 1: XML Familie [30]

1a. Gegeben ist folgendes XML Dokument mit Filminformationen:

```
<?xml version="1.0"?>
<movies>
  <movie id="tt0065214" year="1969">
    <title>The Wild Bunch</title>
    <director>Sam Peckinpah</director>
    <genre>Action</genre>
    <genre>Drama</genre>
    <genre>Western</genre>
  </movie>
  <movie id="tt0083658" year="1982">
    <title>Blade Runner</title>
    <director>Ridley Scott</director>
    <genre>Drama</genre>
    <genre>Sci-Fi</genre>
    <genre>Thriller</genre>
  </movie>
  <movie id="tt0069762" year="1973">
    <title>Badlands</title>
    <director>Terrence Malick</director>
    <genre>Crime</genre>
    <genre>Drama</genre>
    <genre>Romance</genre>
    <genre>Thriller</genre>
  </movie>
  ... weitere Filme ...
</movies>
```

Ihre Aufgabe ist nun, ein oder mehrere XSL Templates zu definieren, mit dem/denen man die obige XML Filmdatenbank in eine HTML Liste mit folgendem Format transformieren kann:

```
<ul>
  <li>Sam Packinpah: <b>The Wild Bunch</b> (1969)</li>
  <li>Ridley Scott: <b>Blade Runner</b> (1982)</li>
  <li>Terrence Malick: <b>Badlands</b> (1973)</li>
  ... weitere Filme ...
</ul>
```

Anmerkungen: Zu jedem Film sollen also der Regisseur (`director`), der Filmtitel (`title`) in Fettschrift und das Jahr (`year`) in Klammer ausgegeben werden. [12]

Aufgabe 1: XML Familie [40]

Gegeben ist folgendes XML Dokument mit Informationen über Spielfilme:

```
<?xml version="1.0"?>
<movies>
  <movie id="tt0065214" year="1969">
    <title>The Wild Bunch</title>
    <director>Sam Peckinpah</director>
    <genre>Action</genre>
    <genre>Drama</genre>
    <genre>Western</genre>
  </movie>
  <movie id="tt0083658" year="1982">
    <title>Blade Runner</title>
    <director>Ridley Scott</director>
    <genre>Drama</genre>
    <genre>Sci-Fi</genre>
    <genre>Thriller</genre>
  </movie>
  <movie id="tt0069762" year="1973">
    <title>Badlands</title>
    <director>Terrence Malick</director>
    <genre>Crime</genre>
    <genre>Drama</genre>
    <genre>Romance</genre>
    <genre>Thriller</genre>
  </movie>

  ... weitere Filme ...

</movies>
```

«MATNR» «NACHNAME» «VORNAME»

1a. Formulieren Sie ein oder mehrere **XSL Templates**, mit dem/denen man die obige XML Filmdatenbank in eine HTML Liste mit folgendem Format transformieren kann:

```
<ul>
  <li><b>The Wild Bunch</b>: Sam Peckinpah, 1969</li>
  <li><b>Blade Runner</b>: Ridley Scott, 1982</li>
  <li><b>Badlands</b>: Terrence Malick, 1973</li>
 ... weitere Filme ...
</ul>
```

Anmerkung: Zu jedem Film sollen also der Filmtitel (`title`) in Fettschrift, der Regisseur (`director`) und das Jahr (`year`) in dem dargestellten Format ausgegeben werden. [12]

1b. Formulieren Sie jeweils einen **XPath** für folgende Abfragen auf die XML Filmdatenbank:

i. Alle Filme vor dem Jahr 2000, die nur einem einzigen Genre zugeordnet sind, und deren Regisseur entweder "Sergio Leone" oder "«VORNAME» «NACHNAME»" war. [3]

ii. Alle Filme, deren Regisseur mehr als einen Film gedreht hat. [3]

1c. Gegeben ist folgende DTD:

```
<!ELEMENT r (x|y|z)>
<!ELEMENT x ((a|b)+|c*)>
<!ELEMENT y (a?,b)+>
<!ELEMENT z (#PCDATA)>
<!ELEMENT a EMPTY>  <!ELEMENT b EMPTY>  <!ELEMENT c EMPTY>
```

In der folgenden Tabelle steht jede Zeile für ein eigenes XML Dokument. Kreuzen Sie in jeder Zeile an, ob das XML *gültig* oder *ungültig* für die gegebene DTD ist! [12]

	gültig	**ungültig**
`<r><y/></r>`		
`<r><z/></r>`		
`<r><x><b/></x></r>`		
`<r><z>Hello, my <dear/></z></r>`		
`<r><x><b/><a/><b/></x></r>`		
`<r><x><c/><c/><c/></x></r>`		
`<r><x><y/></x></r>`		
`<r><x/></r>`		
`<r><y><b/><b/><a/></y></r>`		
`<r><z>Hallo</z></r>`		
`<r><z><a/><b/></z></r>`		
`<r><x><a/><c/></x></r>`		
`<r><y><b/><a/></y></r>`		
`<r><y>zwei noch</y></r>`		
`<r><y><b/><b/></y></r>`		
`<r><x><a nice="day"/></x></r>`		

1d. Erläutern Sie anhand einer kleinen XML Beispieldatei den Zweck und das Konzept von **XML Namespaces** (Anmerkung: nicht nur das XML niederschreiben, sondern auch in eigenen Worten erläutern!) [10]

Aufgabe 1: XML Familie [38]

1a. Gegeben ist folgende DTD:

```
<!ELEMENT A (B+,C?,D*)>
<!ELEMENT B (E*)>
<!ELEMENT E EMPTY>
<!ELEMENT C (#PCDATA)>
<!ELEMENT D (#PCDATA)>
<!ATTLIST B x ID #REQUIRED y (1|2|3) #REQUIRED>
<!ATTLIST E z CDATA #REQUIRED>
<!ATTLIST C r IDREFS #REQUIRED>
<!ATTLIST D s NMTOKEN #REQUIRED>
```

Erstellen Sie ein für diese DTD gültiges XML Dokument unter folgenden verbalen Zusatzeinschränkungen:

- Die Elemente `B`, `D` und `E` müssen jeweils mindestens zweimal vorkommen.
- Das Element `C` muss vorkommen.
- Schreiben Sie in jedes Element einen Textinhalt, sofern es die DTD erlaubt.
- Es dürfen keine leeren Attributwerte vorkommen. [10]

«MATNR» «NACHNAME» «VORNAME»

1b. Beschreiben Sie verbal jeweils kurz folgende Bestandteile der logischen Struktur von XML. [12]

- Wurzelelement:

- Element:

- Elementinhalt:

- Tag:

- XML Deklaration:

- Textknoten:

- Attribut:

«MATNR» «NACHNAME» «VORNAME»

1c. XPath und Achsen: In der Tabelle unten sehen Sie 8-mal das gleiche XML Dokument. Der grau hinterlegte Knoten sei jeweils der Kontextknoten für den XPath Ausdruck, der über dem XML Dokument steht. Markieren Sie jeweils im XML Dokument die Ergebnisknoten des darüberstehenden XPath Ausdrucks (z.B. durch Einkreisen der einzelnen Ergebnisknoten). [8]

`child::d`
 	
`<a>`
` <b/>`
` <b>`
`  <c/>`
`  <c>`
`  <d/>`
`  <d/>`
`  </c>`
`  <c/>`
` </b>`
` <b>`
`  <c/>`
` </b>`
` <b/>`
` <b/>`
`</a>`
	`preceding::*`
	
	`<a>`
	` <b/>`
	` <b>`
	`  <c/>`
	`  <c>`
	`  <d/>`
	`  <d/>`
	`  </c>`
	`  <c/>`
	` </b>`
	` <b>`
	`  <c/>`
	` </b>`
	` <b/>`
	` <b/>`
	`</a>`
		`../descendant::*`
		
		`<a>`
		` <b/>`
		` <b>`
		`  <c/>`
		`  <c>`
		`  <d/>`
		`  <d/>`
		`  </c>`
		`  <c/>`
		` </b>`
		` <b>`
		`  <c/>`
		` </b>`
		` <b/>`
		` <b/>`
		`</a>`
			`ancestor-or-self::*`
			
			`<a>`
			` <b/>`
			` <b>`
			`  <c/>`
			`  <c>`
			`  <d/>`
			`  <d/>`
			`  </c>`
			`  <c/>`
			` </b>`
			` <b>`
			`  <c/>`
			` </b>`
			` <b/>`
			` <b/>`
			`</a>`

`preceding-sibling::*`
`<a>`
` <b/>`
` <b>`
`  <c/>`
`  <c>`
`  <d/>`
`  <d/>`
`  </c>`
`  <c/>`
` </b>`
` <b>`
`  <c/>`
` </b>`
` <b/>`
` <b/>`
`</a>`
	`following::*`
	
	`<a>`
	` <b/>`
	` <b>`
	`  <c/>`
	`  <c>`
	`  <d/>`
	`  <d/>`
	`  </c>`
	`  <c/>`
	` </b>`
	` <b>`
	`  <c/>`
	` </b>`
	` <b/>`
	` <b/>`
	`</a>`
		`parent::*`
		
		`<a>`
		` <b/>`
		` <b>`
		`  <c/>`
		`  <c>`
		`  <d/>`
		`  <d/>`
		`  </c>`
		`  <c/>`
		` </b>`
		` <b>`
		`  <c/>`
		` </b>`
		` <b/>`
		` <b/>`
		`</a>`
			`/*`
			
			`<a>`
			` <b/>`
			` <b>`
			`  <c/>`
			`  <c>`
			`  <d/>`
			`  <d/>`
			`  </c>`
			`  <c/>`
			` </b>`
			` <b>`
			`  <c/>`
			` </b>`
			` <b/>`
			` <b/>`
			`</a>`

1d. Erläutern Sie anhand eines kleinen selbstgewählten XML Ausschnitts den Zweck und das Konzept von **XML Namespaces** (Anmerkung: nicht nur das XML niederschreiben, sondern auch in eigenen Worten erläutern!) [8]

Aufgabe 1: XML Technologien [38]

Gegeben ist folgende XML Datei:

```
<?xml version="1.0" encoding="utf-8"?>
<ehepaare>
  <ehepaar hochzeitsTag="1983-01-22">
    <scheidung>2001-12-24</scheidung>
    <mann svnr="2948120145">Karli Zwiefel</mann>
    <frau svnr="1234050342">Mitzi Knofel</frau>
    <standesamt>Wien 9</standesamt>
  </ehepaar>
  <ehepaar hochzeitsTag="2002-08-12">
    <frau svnr="1234050342">Mitzi Knofel</frau>
    <mann svnr="3203070382">Franz Radiwurzn</mann>
    <standesamt>St. Pölten</standesamt>
  </ehepaar>
  <ehepaar hochzeitsTag="1954-09-02">
    <standesamt>Los Angeles</standesamt>
    <frau svnr="4419021232">Jane Doe</frau>
    <mann svnr="1934081220">Henry Chinaski</mann>
    <scheidung>1957-01-01</scheidung>
  </ehepaar>
  <ehepaar hochzeitsTag="1998-02-02">
    <mann svnr="4756260268">Ihsan Farha</mann>
    <frau svnr="3487010157">Chip Stack</frau>
    <standesamt>Las Vegas</standesamt>
  </ehepaar>

  ... weitere Ehepaare ...

</ehepaare>
```

1a. XPath. Formulieren Sie jeweils einen XPath für folgende Abfragen auf die obige XML Datei:

i. Die Anzahl der am Standesamt St. Pölten geschlossenen Ehen. [2]

ii. Alle Standesämter, an denen bereits Ehen geschlossen wurden, jedoch nur solche, die nicht geschieden wurden. [4]

iii. Die Namen aller Frauen, die bereits mehrmals verheiratet waren. Anmerkung: Die Sozialversicherungen (Attribut `svnr`) seien eindeutig. [4]

iv. Die Scheidungsrate (also das Verhältnis von geschiedenen Ehen zu nicht geschiedenen Ehen). [3]

1b. XSLT. Sie wollen nun die obige XML Datei nach HTML transformieren, sodass der Output wie im folgenden Ausschnitt angedeutet aussehen soll: [12]

```
<ol>
  <li>Karli Zwiefel heiratet Mitzi Knofel am 1983-01-22 im Standesamt Wien 9.</li>
  <li>Franz Radiwurzn heiratet Mitzi Knofel am 2002-08-12 im Standesamt St. Pölten.</li>
  ... und so weiter ...
</ol>
```

Fügen Sie das benötigte Template in folgendes Gerüst ein:

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="html"/>









</xsl:stylesheet>
```

1c. XML Schema. Komplettieren Sie die grau unterlegten Teile des unten stehenden Schemas für die gegebene XML Datei, wobei neben den in der XML Datei ersichtlichen Einschränkungen folgende Zusatzangaben gelten. [13]

- Die Sozialversicherungsnummer (Attribut `svnr`) ist ein String mit genau 10 Zeichen.
- Der Hochzeitstag (Attribut `hochzeitsTag`) ist ein Datum.
- Es gibt keine optionalen Attribute.

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema xmlns:      ="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="ehepaare" type="wurzelType"/>

  <xsd:complexType name="wurzelType">
    <xsd:sequence>
      <xsd:element name="          " type="ehepaarType" maxOccurs="          "/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="personType">
    <xsd:                    >
      <xsd:extension base="xsd:string">
        <xsd:attribute type="svnrType"                                  />
      </xsd:extension>
    </xsd:                   >
  </xsd:complexType>

  <xsd:simpleType name="svnrType">
    
  </xsd:simpleType>

  <xsd:complexType name="ehepaarType">
    
  </xsd:complexType>
</xsd:schema>
```

Aufgabe 1: XML Technologien [38]

1a. Logische Struktur von XML. Beschreiben Sie so umfassend wie möglich folgende Bestandteile der logischen Struktur von XML. (Stellen Sie sich z.B. jeweils folgende Fragen: Was ist das? Wo befindet es sich in der XML Datei? Wie sieht es aus? Welche Bestandteile hat es bzw. kann es haben? Welche Ausprägungen kann es annehmen? etc.) [12]

- Wurzelelement:

- Element:

- Elementinhalt:

- Tag:

- XML Deklaration:

- Attribut:

1b. XSLT. Gegeben sind jeweils eine XML Datei und ein Stylesheet. Notieren Sie jeweils die Ausgabe bei Anwendung des Stylesheet auf die XML Datei. [14] = [2+4+8]

XML Datei:

```
<buch>
 <autor>C. Bukowski</autor>
 <titel>Hollywood</titel>
</buch>
```

XSL Stylesheet:

```
<xsl:template match="autor">
 <xsl:value-of select="text()"/>
</xsl:template>
```

Ausgabe:

XML Datei:

```
<buch>
 <autor>
   <titel>Dr.</titel>
   <vname>«VORNAME»</vname>
   <nname>«NACHNAME»</nname>
 </autor>
 <titel>Bring 'em on!</titel>
</buch>
```

XSL Stylesheet:

```
<xsl:template match="autor">
 Vorname: <xsl:value-of select="vname"/>
</xsl:template>
<xsl:template match="titel">
 Titel: <xsl:value-of select="."/>
</xsl:template>
<xsl:template match="nname">
 Nachname: <xsl:value-of select="nname"/>
</xsl:template>
```

Ausgabe:

XML Datei:

```
<A>
  <B a="47" b="«MATNR»">
   <C>x</C>
   <C t="woa">y</C>
   <C><D>muh</D><D/></C>
 </B>
 <B a="«NACHNAME»">
   <D>blah</D>
   <D k="1" r="3">dah</D>
 </B>
 <D>jabba</D>
</A>
```

XSL Stylesheet:

```
<xsl:template match="B[@*]">
  B: <xsl:for-each select="@*">
    <xsl:value-of select="."/> /
  </xsl:for-each>
  Mehr: <xsl:apply-templates select="*/*"/>
</xsl:template>

<xsl:template match="D">
  D = <xsl:value-of select="text()"/> !!!
</xsl:template>
```

Ausgabe:

«MATNR» «NACHNAME» «VORNAME»

1c. DTD. Gegeben ist folgende DTD:

```
<!ELEMENT m (k|x|z)>
<!ELEMENT k (s?,b)+>
<!ELEMENT x ((s|b)+|c*)>
<!ELEMENT z (#PCDATA)>
<!ELEMENT s EMPTY>
<!ELEMENT b EMPTY>
<!ELEMENT c EMPTY>
```

In der folgenden Tabelle steht jede Zeile für ein eigenes XML Dokument. Kreuzen Sie in jeder Zeile an, ob das XML *gültig* oder *ungültig* für die gegebene DTD ist! [12]

	gültig	**ungültig**
`<m><x><b/></x></m>`		
`<m><z>Sapper <lott/></z></m>`		
`<m><x><b/><s/><b/></x></m>`		
`<m><x><c/><c/><c/></x></m>`		
`<m><x><k/></x></m>`		
`<m><k><b/><b/></k></m>`		
`<m><z><s/><b/></z></m>`		
`<m><k>David Banner</k></m>`		
`<m><k><b/><s/></k></m>`		
`<m><x><s cool="night"/></x></m>`		
`<m><k/></m>`		
`<m><z>Hans</z></m>`		
`<m><k><b/><b/><s/></k></m>`		
`<m><x><s/><c/></x></m>`		

«MATNR» «NACHNAME» «VORNAME»

Aufgabe 1: XML Technologien [41]

Gegeben ist folgendes XML Schema. (Hinweis: die Zahlen am Beginn jeder Zeile sind Zeilennummern, auf die Sie sich bei der Lösung der Aufgabe beziehen können.)

```
 1 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 2
 3  <xsd:complexType name="wohnungen_t">
 4   <xsd:sequence minOccurs="2" maxOccurs="unbounded">
 5    <xsd:element name="wohnung" type="wohnung_t" minOccurs="0"/>
 6    <xsd:element name="makler" type="xsd:string" minOccurs="0"/>
 7   </xsd:sequence>
 8  </xsd:complexType>
 9
10  <xsd:complexType name="wohnung_t">
11   <xsd:sequence>
12    <xsd:element name="eigentuemer" type="eigentuemer_t"/>
13    <xsd:element name="mieter" type="xsd:string"/>
14    <xsd:element name="typ" type="typ_t"/>
15   </xsd:sequence>
16   <xsd:attribute name="ort" type="xsd:string" use="required"/>
17   <xsd:attribute name="flaeche" type="xsd:positiveInteger" use="required"/>
18   <xsd:attribute name="info" type="xsd:string" use="optional"/>
19  </xsd:complexType>
20
21  <xsd:simpleType name="eigentuemer_t">
22   <xsd:restriction base="xsd:string">
23    <xsd:maxLength value="10"/>
24   </xsd:restriction>
25  </xsd:simpleType>
26
27  <xsd:simpleType name="typ_t">
28   <xsd:restriction base="xsd:string">
29    <xsd:enumeration value="Miete"/>
30    <xsd:enumeration value="Eigentum"/>
31    <xsd:enumeration value="Geschäft"/>
32    <xsd:enumeration value="Pacht"/>
33   </xsd:restriction>
34  </xsd:simpleType>
35
36  <xsd:element name="wohnungen" type="wohnungen_t">
37   <xsd:unique name="x">
38    <xsd:selector xpath="wohnung"/>
39    <xsd:field xpath="eigentuemer"/>
40   </xsd:unique>
41  </xsd:element>
42
43 </xsd:schema>
```

«MATNR» «NACHNAME» «VORNAME»

1a. Schema-Validierung. Markieren Sie jene Stellen im folgenden XML Dokument, die in Bezug auf das gegebene XML Schema *ungültig* sind. Begründen Sie jeweils kurz, warum die Stelle ungültig ist. [15]

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <wohnungen>
4    <makler>Immobilienbüro Dr. Rohdiamant</makler>
5
6    <wohnung ort="Gols" info="möbliert">
7      <eigentuemer>«VORNAME» «NACHNAME»</eigentuemer>
8      <mieter>-</mieter>
9      <typ>Geschäft</typ>
10     <typ>Pacht</typ>
11     <zimmer>7</zimmer>
12   </wohnung>
13
14   <wohnung ort="Wien 17" flaeche="92">
15     <eigentuemer>M. Derntl</eigentuemer>
16     <mieter>H. Reisinger</mieter>
17     <typ>Miete Geschäft</typ>
18   </wohnung>
19
20   <wohnung ort="Linz" flaeche="152" info="Altbaujuwel"/>
21
22   <wohnung ort="Graz" wohnflaeche="68">
23     <eigentuemer>A. Schwarzenegger</eigentuemer>
24     <typ>Geschäft</typ>
25     <mieter/>
26   </wohnung>
27
28   <wohnung ort="Wien 21" flaeche="98m²">
29     <eigentuemer>M. Derntl</eigentuemer>
30     <mieter>H. Hölzl</mieter>
31     <typ>Pacht</typ>
32   </wohnung>
33 </wohnungen>
```

1b. Schema vs. DTD. Erklären Sie, welche in dem gegebenen XML Schema definierten Einschränkungen Sie mit einer DTD nicht definieren könnten (mit kurzer Begründung!) [6]

1c. XPath. Nehmen Sie an, es existiert ein XML Dokument, das gültig für das gegebene XML Schema ist. Erstellen Sie nun je einen XPath Ausdruck für folgende Abfragen:

i. Die Anzahl der Wohnungen mit Typ "Eigentum" außerhalb Wiens und einer Fläche von mindestens 85. [3]

ii. Die Mieter aller Wohnungen, deren Eigentümer "«VORNAME» «NACHNAME»" heißt. [2]

iii. Alle Orte, in denen es mindestens zwei Wohnungen vom Typ „Miete" gibt. (Hinweis: dafür brauchen Sie XPath Achsen!) [3]

«MATNR» «NACHNAME» «VORNAME»

1d. XSL Transformation. Gegeben ist folgende DTD:

```
<!ELEMENT personen (person*)>
<!ELEMENT person (email?, homepage*)>
<!ATTLIST person id ID #REQUIRED vorname CDATA #REQUIRED nachname CDATA #REQUIRED>
<!ELEMENT email (#PCDATA)>
<!ELEMENT homepage (#PCDATA)>
<!ATTLIST homepage typ (privat|beruflich) "privat">
```

Gesucht ist ein XSL-Stylesheet, das eine ungeordnete HTML-Liste (`<UL>`) aller Personen (Vor- und Nachname) ausgibt, die über eine E-Mail-Adresse verfügen. Zu jeder Person soll auch in Klammer die E-Mail-Adresse (als Hyperlink mit `mailto:`) ausgegeben werden. [12]

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
  <xsl:output method="html"/>









</xsl:stylesheet>
```

«MATNR» «NACHNAME» «VORNAME»

Aufgabe 1: XML Technologien [40]

Gegeben ist folgendes XML Schema. (Hinweis: die Zahlen am Beginn jeder Zeile sind Zeilennummern, auf die Sie sich bei der Lösung der Aufgabe beziehen können.)

```
 1 <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 2
 3   <xsd:complexType name="banken_t">
 4     <xsd:sequence maxOccurs="1">
 5       <xsd:element name="bank" type="bank_t" maxOccurs="unbounded"/>
 6       <xsd:element name="CEO" type="xsd:string"/>
 7     </xsd:sequence>
 8     <xsd:attribute name="holding" type="xsd:string" use="required"/>
 9   </xsd:complexType>
10
11   <xsd:complexType name="bank_t">
12     <xsd:sequence>
13       <xsd:element name="chef" type="chef_t"/>
14       <xsd:element name="gründungsjahr" type="xsd:gYear"/>
15       <xsd:element name="kunden" type="kunden_t"/>
16       <xsd:element name="wertpapierkennung" type="wpk_t" minOccurs="0"/>
17     </xsd:sequence>
18     <xsd:attribute name="name" type="xsd:string" use="required"/>
19     <xsd:attribute name="ort" type="xsd:string" use="required"/>
20     <xsd:attribute name="land" type="xsd:string" use="optional"/>
21   </xsd:complexType>
22
23   <xsd:simpleType name="chef_t">
24     <xsd:restriction base="xsd:string">
25       <xsd:maxLength value="10"/>
26     </xsd:restriction>
27   </xsd:simpleType>
28
29   <xsd:complexType name="kunden_t">
30     <xsd:attribute name="firmen" type="xsd:nonNegativeInteger" use="required"/>
31     <xsd:attribute name="privat" type="xsd:nonNegativeInteger" use="required"/>
32   </xsd:complexType>
33
34   <xsd:simpleType name="wpk_t">
35     <xsd:restriction base="xsd:string">
36       <xsd:length value="12"/>
37     </xsd:restriction>
38   </xsd:simpleType>
39
40   <xsd:element name="banken" type="banken_t">
41     <xsd:unique name="xxx">
42       <xsd:selector xpath="bank"/>
43       <xsd:field xpath="chef"/>
44     </xsd:unique>
45   </xsd:element>
46
47 </xsd:schema>
```

«MATNR» «NACHNAME» «VORNAME»

1a. Schema-Validierung. Markieren Sie jene Stellen im folgenden XML Dokument, die in Bezug auf das gegebene XML Schema *ungültig* sind. Begründen Sie jeweils kurz, warum die Stelle ungültig ist. [15]

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <banken holding="UniCredit Group">
4
5    <CEO>Gennaro Gattuso</CEO>
6
7    <bank name="Bank Pekao SA" ort="Warschau" land="Polen">
8      <chef>«VORNAME» «NACHNAME»</chef>
9      <gründungsjahr>1929</gründungsjahr>
10     <kunden privat="299576" firmen="2434">
11     </kunden>
12     <wertpapierkennung>123456789012</wertpapierkennung>
13   </bank>
14
15   <bank name="HypoVereinsbank" ort="München" land="">
16     <chef>K. Amel</chef>
17     <kunden firmen="344654" privat=""/>
18     <gründungsjahr>-</gründungsjahr>
19     <wertpapierkennung/>
20   </bank>
21
22   <bank name="Bank Ruptcy, Inc." ort="Boston, USA">
23   </bank>
24
25   <bank name="Zagrebacka banka" land="Kroatien" ort="Zagreb">
26     <chef>K. Amel</chef>
27     <kunden firmen="keine" privat="398476"/>
28     <gründungsjahr></gründungsjahr>
29   </bank>
30
31 </banken>
```

1b. Schema vs. DTD. Erklären Sie, welche in dem oben gegebenen XML Schema definierten Einschränkungen Sie mit einer DTD nicht definieren könnten (beziehen Sie sich auf Zeilennummern im Schema und geben Sie jeweils eine kurze Begründung!) [8]

1c. XPath. Nehmen Sie an, es existiert ein XML Dokument, das gültig für das gegebene XML Schema ist. Erstellen Sie nun je einen XPath Ausdruck für folgende Abfragen:

i. Die Anzahl der Banken im Land Österreich mit mindestens 2500 Firmenkunden? [3]

ii. Die Orte aller Banken, deren Chef "«VORNAME» «NACHNAME»" heißt? [2]

iii. Die Jahre, in denen mindestens zwei Banken in Frankreich gegründet wurden? [4]

«MATNR» «NACHNAME» «VORNAME»

1d. XSL Transformation. Notieren Sie die Ausgabe bei der Anwendung des XSL Stylesheet (rechts) auf die gegebene XML Datei (links). [8]

XML Datei:

```
<?xml version="1.0"?>
<X>
  <Y a="«NACHNAME»">
   <R>Kupfer</R>
   <R n="1" m="3">Eisen</R>
  </Y>
  <R>Holz</R>
  <Y x="47" y="«MATNR»">
   <Z>Gold</Z>
   <Z><R>Gummi</R><R/></Z>
   <Z t="woa">Nickel</Z>
</Y>
</X>
```

XSL Stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="Y[@*]">
    Y: <xsl:for-each select="@*">
      <xsl:value-of select="."/> /
    </xsl:for-each>
    Mehr: <xsl:apply-templates select="*/*"/>
  </xsl:template>

  <xsl:template match="R">
    R = <xsl:value-of select="text()"/> !!!
  </xsl:template>

</xsl:stylesheet>
```

Ausgabe: