Klausur "Semistrukturierte Daten und XML"
Sommersemester 2006
Prof. Dr. Wolfgang May
2. August 2006, 10-12 Uhr
Bearbeitungszeit: 90 Minuten

Vorname:

Nachname:

Matrikelnummer:

Bei der Klausur sind **keine Hilfsmittel** (Skripten, Taschenrechner etc.) erlaubt. Handies müssen ausgeschaltet sein. Papier wird gestellt. Benutzen Sie nur die **ausgeteilten**, zusammengehefteten **Blätter** für Ihre Antworten. Schreiben Sie mit blauem/schwarzem Kugelschreiber, Füller etc.; Bleistift ist nicht erlaubt. Beantworten Sie die Fragen auf Deutsch oder Englisch.
Auf dem letzten Blatt finden Sie ein XML-Dokument, das in den Aufgaben 2 – 4 verwendet wird. Trennen Sie es ggf. zur Bearbeitung der Aufgaben ab.

Zum **Bestehen** der Klausur sind **45** Punkte hinreichend.

☐ meine Note soll mit Matrikelnummer so bald wie möglich auf der Vorlesungs-Webseite veröffentlicht werden.

☐ meine Note soll nicht veröffentlicht werden; ich erfahre sie dann aus Munopag (bzw. für nicht im Munopag geführte Studierende: beim Abholen des Scheins).

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (Allgemeines)	9	4
Aufgabe 2 (DTD, Validierung)	34+4	20
Aufgabe 3 (XML, XPath, XQuery)	22	12
Aufgabe 4 (XSLT)	25	13
Summe	90+4	49

Note:

Aufgabe 1 (Allgemeines [9 Punkte])

1. Beschreiben Sie kurz, aus welchen Komponenten ein *Lightweight Object* in dem in der Vorlesung besprochenen frühen semistrukturierten Datenmodell "OEM - Object Exchange Model" besteht. (2 P)

2. Stellen Sie grafisch dar, wie dabei eine Person mit Namen "John Doe" und Geburtsdatum 1.1.1990 in OEM dargestellt wird (3 P)

3. Geben Sie an, wie dieselbe Information im relationalen Modell dargestellt wird (2 P).

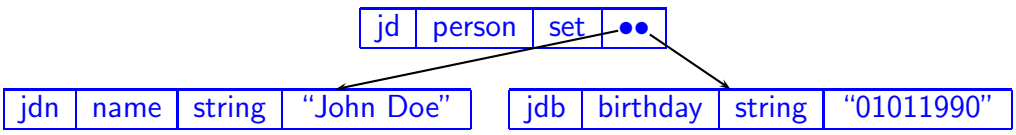
4. Geben Sie weiterhin an, wie dieselbe Information in XML dargestellt werden kann (2 P).

Lösung

1. Es hat 4 Komponenten: Objekt-ID, Label, Datentyp, Wert.

2. <jd, person, set, {jdn, jdb}>,
   <jdn, name, string, "John Doe">,
   <jdb, birthday, string, "01011990">.

   Oder als Grafik:



3.

Person	
Name	Geburtsdatum
"John Doe"	01-01-1990

4. <person name="John Doe" geburtsdatum="01011990" />
   oder

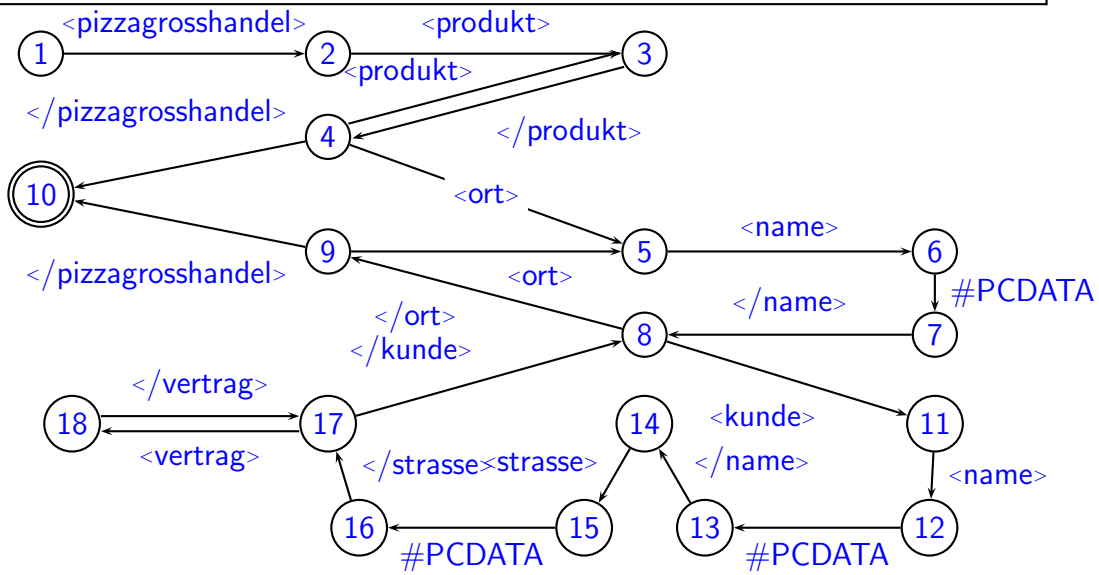
```
<person>
    <name>John Doe</name>
    <geburtsdatum>01011990</geburtsdatum>
</person>
```

Aufgabe 2 (DTD, Validierung [34+4 Punkte])
Diese Aufgabe verwendet die auf dem hintersten Blatt zu findende Pizzagrosshandel-Datenbasis.

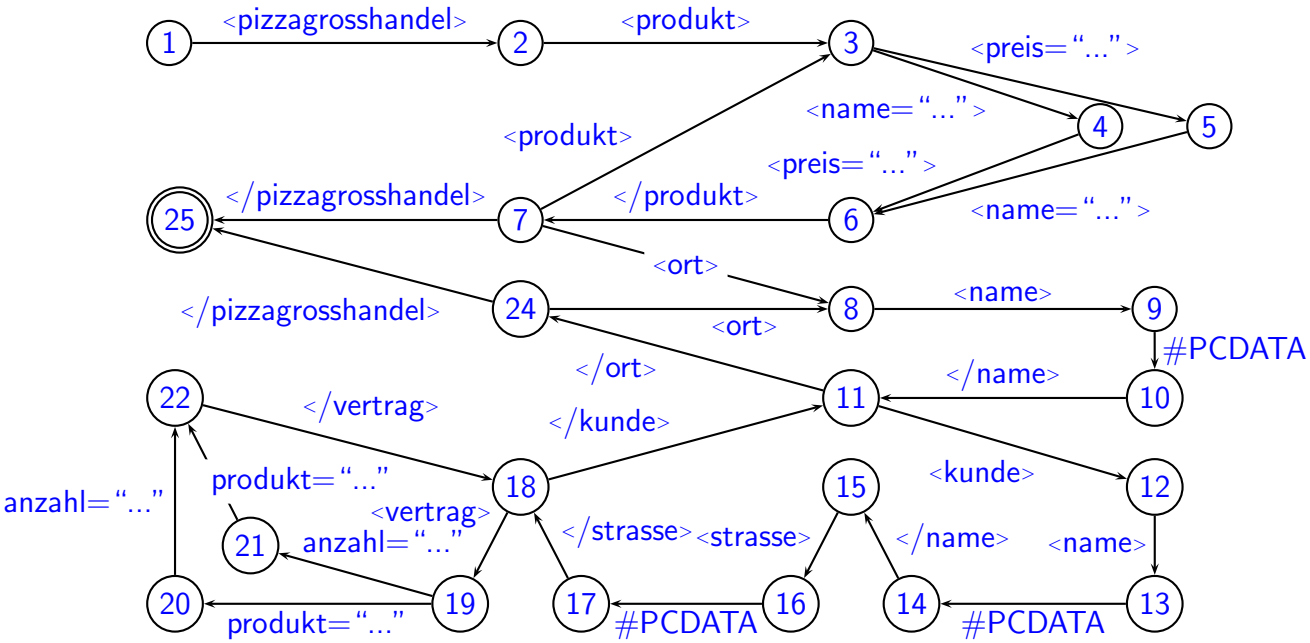
1. Geben Sie eine DTD zu der Datenbasis an. Entscheiden Sie bei dem Entwurf, welche Attribute welcher Elemente Sie als ID auszeichnen würden (12 P). (Diese dürfen Sie im folgenden auch als IDs verwenden)
2. Geben Sie einen endlichen Automaten (als Diagramm) zur DTD an, mit dem sich ein Dokument zu dieser DTD validieren läßt. Sie dürfen dabei Attribute ignorieren (nehmen Sie an, dass diese bei dem Übergang, der den öffnenden Tag verarbeitet, überprüft werden) (12 P).
3. Welche in der DTD enthaltenen Anforderungen an ein solches Dokument können mit einem endlichen Automaten nicht geprüft werden? (2 P)
4. Können Sie theoretisch begründen, warum dies nicht gehen kann? (bis zu 4 Zusatzpunkte)
5. Wie können Sie die Vorgehensweise bei der Validation so erweitern, dass diese Anforderungen ebenfalls geprüft werden? (8 P)

Lösung

1.
```
<!ELEMENT pizzagrosshandel (produkt+,ort*)>
<!ELEMENT produkt EMPTY>
  <!ATTLIST produkt name ID #REQUIRED
                    preis CDATA #REQUIRED>
<!ELEMENT ort (name,kunde*)>
<!ELEMENT kunde (name,strasse,vertrag*)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT strasse (#PCDATA)>
<!ELEMENT vertrag EMPTY>
  <!ATTLIST vertrag produkt IDREF #REQUIRED
                    anzahl CDATA #REQUIRED>
```

2.


... und hier ein Automat, der auch die Attribute enthält:



3. Die Werte von ID-Attributen müssen *dokumentweit* eindeutig sein. Die Werte von IDREF- (und IDREFS-) Attributen müssen als Werte von ID-Attributen im Dokument vorhanden sein.

4. endliche Automaten können reguläre Sprachen parsen. Jede einzelne Elementdeklaration einer DTD ist regulär und kann von einem endlichen Automaten validiert werden. Bei nicht-rekursiven DTDs kann man diese Automaten zu einem großen Automaten zusammensetzen.

   Bei rekursiven muss man einen Stack hinzunehmen, um die Klammerungstiefe zu überwachen. Alternativ kann man auch zu Kellerautomaten greifen (wobei der Keller aber eben nur zur Überwachung der Schachtelung genutzt wird – die volle Ausdruckskraft kontextfreier Sprachen wird nicht benötigt).

   Die *strukturellen* Elemente einer DTD sind damit im allgemeinen Fall kontextfrei.

   Das Überprüfen der Eindeutigkeit von ID sowie des Vorhandenseins von IDs ist nicht kontextfrei, da eine Art "Negation" (Eindeutigkeit!) benötigt wird – siehe Pumping Lemma. Dasselbe gilt übrigens auch für Sprachen wie Pascal, C, C++, Java: die Grundstruktur ist kontextfrei, aber die Überprüfung von Variablen- und Prozedur-/Funktions- bzw. Methodendeklarationen nicht.

   Man benötigt daher weitere Datenstrukturen und muss diese im Validierungsalgorithmus verwenden.

5. IDs und IDREFs zusätzlich verwalten:

- In dem Automaten werden bei der Erstellung alle Attribute, die ID oder IDREF sind, markiert.
- ein leeres Dictionary (Suchbaum, Hash, etc) für ID-Werte sowie eine leere Liste für IDREF-Werte erzeugen.
- Dokument mit dem Automaten durchgehen. Jedesmal wenn ein ID-Attribut verarbeitet wird, dieses in das ID-Dictionary einfügen. Wenn es bereits existiert, Fehler. Jedesmal wenn ein IDREF/IDREFS-Attribut verarbeitet wird, dieses in die IDREF-Liste einfügen (nicht sofort überprüfen, ob es existiert, da es Vorwärtsreferenzen geben kann).
- jetzt für jeden IDREF-Wert in der Liste überprüfen, ob er in dem ID-Dictionary enthalten ist.

Aufgabe 3 (XML, XPath, XQuery [22 Punkte])
Diese Aufgabe verwendet die auf dem hintersten Blatt zu findende Pizzagrosshandel-Datenbasis.
Geben Sie für die Aufgabenteile (1)-(5) je eine XPath- oder XQuery-Anfrage an, die das Ergebnis in dem jeweils angegebenen Format ausgibt (falls kein Format angegeben, beliebig).

1. Geben Sie eine XPath- oder XQuery-Anfrage an, die die Namen aller Pizzerien ergibt, die Lasagne geliefert bekommen (1 P).

   Lösung

   ```
   //kunde[vertrag/@produkt='Lasagne']/name/text()
   ```

2. Geben Sie eine XPath- oder XQuery-Anfrage an, die die Menge aller Paare (Ort, Produkt) ergibt, so dass dieses Produkt an den angegebenen Ort geliefert wird (3 P).
   Ausgabeformat: <answer ort='*ortsname*' produkt='*produktname*'/>

   Lösung Wichtig ist dabei im wesentlichen, Duplikate zu vermeiden.
   Man kann dies durch ein Join beider Wertebereiche mit where-Klausel tun:

   ```
   for $ort in //ort, $p in //produkt
   where $ort/kunde/vertrag[@produkt = $p/@name]
   return
   <answer ort='{$ort/name}' produkt='{$p/@name}'/>
   ```

   Alternative: Korrelierte Variablen und explizite Duplikateliminierung:

   ```
   for $ort in //ort,
         $p in distinct-values($ort/vertrag/@produkt)
   return
   <answer ort='{$ort/name}' produkt='{$p}'/>
   ```

   Duplikateliminierung durch Anwendung der ID-Funktion:

   ```
   for $ort in //ort,
         $p in id($ort/vertrag/@produkt)
   return
   <answer ort='{$ort/name}' produkt='{$p/@name}'/>
   ```

   oder die Join-Bedingung in die for-Klausel nehmen:

   ```
   for $p in //produkt,
         $ort in //ort[./kunde/vertrag[@produkt=$p/@name]]
   return
   <answer ort='{$ort/name}' produkt='{$p/@name}'/>
   ```

3. Geben Sie eine XPath- oder XQuery-Anfrage an, die für jede Stadt angibt, wieviele Portionen von jedem Produkt *insgesamt* in diese Stadt geliefert werden (6 P).
Ausgabeformat:

<ort name='*ortsname*'>
  <produkt name='*produktname*$_1$' portionen='*anzahl*$_1$'/>
  <produkt name='*produktname*$_2$' portionen='*anzahl*$_2$'/>
    :
</ort>

Produkte die in die jeweilige Stadt nicht geliefert werden, sollen nicht aufgezählt werden.

Lösung

```
for $ort in //ort
 return
 <ort name='{$ort/name}'>
 { for $p in //produkt
    let $n := sum($ort//vertrag[@produkt=$p/@name]/@anzahl)
  where $n > 0
    return <produkt name='{$p/@name}' portionen='{$n}'/> }
 </ort>
```

4. Geben Sie eine XQuery-Anfrage an, die alle Namen und Orte der Pizzerien ergibt, die keinen Salat geliefert bekommen (4 P).

Lösung Wichtig: für die Negation mengenwertige Semantik von Pfaden beachten!

```
for $kunde in //kunde[not (vertrag/@produkt='Salat')]
return
<answer kunde='{$kunde/name}' ort='{$kunde/ancestor::ort/name}'/>
```

oder

```
for $kunde in //kunde
where not ($kunde/vertrag/@produkt='Salat')
return
<answer kunde='{$kunde/name}' ort='{$kunde/ancestor::ort/name}'/>
```

oder ganz explizit:

```
for $kunde in //kunde
where every $v in $kunde/vertrag satisfies ($v/@produkt !='Salat')
return
<answer kunde='{$kunde/name}' ort='{$kunde/ancestor::ort/name}'/>
```

Wer den Weg vom Kunden zum ancestor-Ort nicht findet, kann es auch anders machen (vgl. Umschreiben von Rückwärtsachsen in Vorwärtsachsen):

```
for $ort in //ort, $kunde in $ort/kunde[not (vertrag/@produkt='Salat')]
return
<answer kunde='{$kunde/name}' ort='{$ort/name}'/>
```

Hinweis: beachten Sie den Unterschied zu der Anfrage "Namen aller Pizzerien, die einen Vertrag über etwas haben, das kein Salat ist".

5. Geben Sie die Namen aller Kunden an, deren Lieferung *alle Produkte umfasst, die teurer als 4 E sind* enthält (8 P).

Lösung Dies ist eine relationale Division. Erste Version: Vorgehensweise wie bei "not exists where not in" von SQL:

```
let $incomplete :=
   (for $p in //produkt[@preis > 4], $k in //kunde
   where not (string($p/@name) =  $k/vertrag/string(@produkt))
   return $k)
(: incomplete contains duplicates :)
for $k in //kunde
 where not ($k/name = $incomplete/name)
return $k/name
```

Mit every geht es deutlich einfacher:

```
for $k in //kunde
where every $p in //produkt[@preis > 4]
     satisfies $p/@name = $k/vertrag/@produkt
return $k/name
```

oder in XPath 2.0:

```
//kunde[every $p in //produkt[@preis > 4] satisfies
          $p/@name = vertrag/@produkt]/name
```

Hinweis: beachten Sie den Unterschied zu den (deutlich einfacheren) Anfragen

- Namen aller Kunden, deren Lieferung *ausschliesslich Produkte umfasst, die teurer als 4 E sind*, sowie
- Namen aller Kunden, deren Lieferung *irgendein Produkt umfasst, das teurer als 4 E ist*.

Aufgabe 4 (XSLT [25 Punkte])
Diese Aufgabe verwendet ebenfalls die auf dem hintersten Blatt zu findende Pizzagrosshandel-Datenbasis.

1. Welche beiden XSL-Elemente bilden die Grundlage jedes Stylesheets? Beschreiben Sie *kurz*, wie diese aussehen, und wie sie zusammenarbeiten. (5 P)

2. Geben Sie ein Stylesheet an, das ein HTML-Dokument erstellt, das für jede zu beliefernde Stadt eine Tabelle enthält, die den Namen der Stadt und zu jeder Adresse die entsprechenden Lieferungen angibt. (d.h., praktisch die Anweisungen an den Auslieferungsfahrer) [20 P]

Lösung Die Hauptelemente sind <xsl:template match="*pattern*"> und <xsl:apply-templates select="*path*">. Jedes Template gibt mit dem match-Attribut an, welche Elemente durch es verarbeitet werden können. Diese Verarbeitung wird durch den Elementinhalt angegeben, der XML-Befehle sowie das zu erzeugende XML (oder auch HTML, oder Text) enthält. In diesem Inhalt können insbesondere apply-templates-Elemente enthalten sein. Für jedes Element, das in der Ergebnismenge ihres select-Attributes enthalten ist, wird das jeweils passende template ausgeführt.

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">
 <xsl:template match="pizzagrosshandel">
   <html> <body>
     <xsl:apply-templates select="ort"/>
   </body> </html>
 </xsl:template>
 <xsl:template match="ort">
   <table>
     <tr><td colspan="2"><b>
       <xsl:value-of select="name"/>
     </b></td></tr>
     <xsl:apply-templates select="kunde"/>
   </table>
 </xsl:template>
 <xsl:template match="kunde">
  <tr><td colspan="2">
    <xsl:value-of select="name"/> ,
    <xsl:value-of select="strasse"/>
  </td></tr>
  <xsl:apply-templates select="vertrag"/>
 </xsl:template>
 <xsl:template match="vertrag">
  <tr><td>
    <xsl:value-of select="@produkt"/>
  </td>
  <td>
    <xsl:value-of select="@anzahl"/>
  </td>
  </tr>
 </xsl:template>
</xsl:stylesheet>
```

Für Aufgaben 3 und 4 (XQuery und XSLT) sei das folgende XML-Dokument, das die Datenbank eines Pizza-Großhandels beschreibt, gegeben: Ein Pizzeria-Großhandel hat Lieferverträge mit Pizzerien, an die er *jede Woche eine bestimmte Anzahl Portionen seiner Produkte* ausliefert.

```
<?xml version="1.0"?>
<!DOCTYPE pizzagrosshandel SYSTEM "pizzagrosshandel.dtd">
<pizzagrosshandel>
  <produkt name="Pizza" preis="5.00"/>
  <produkt name="Lasagne" preis="6.00"/>
  <produkt name="Gnocchi" preis="4.50"/>
  <produkt name="Salat" preis="3.00"/>
  <ort>
    <name>Goettingen</name>
    <kunde>
      <name>Bella Italia</name>
      <strasse>Weender Str. 8</strasse>
      <vertrag produkt="Pizza" anzahl="10"/>
      <vertrag produkt="Lasagne" anzahl="15"/>
      <vertrag produkt="Gnocchi" anzahl="15"/>
      <vertrag produkt="Salat" anzahl="20"/>
    </kunde>
    <kunde>
      <name>Da MafIA</name>
      <strasse>Lotzestrasse 16-18</strasse>
      <vertrag produkt="Gnocchi" anzahl="10"/>
    </kunde>
  </ort>
  <ort>
    <name>Kassel</name>
    <kunde>
      <name>Casino Grande</name>
      <strasse>Am Weinberg 14</strasse>
      <vertrag produkt="Pizza" anzahl="12"/>
      <vertrag produkt="Salat" anzahl="15"/>
    </kunde>
    <kunde>
      <name>Venezia</name>
      <strasse>Koenigsstrasse 111</strasse>
      <vertrag produkt="Pizza" anzahl="20"/>
    </kunde>
  </ort>
</pizzagrosshandel>
```