Example 12.8: The XPath query:

```
/Movies/Movie/Version[Star]
```

applied to the document of Fig. 12.3 returns three `Version` elements. The condition `[Star]` is interpreted as "has at least one `Star` subelement." That condition is true for the `Version` elements of lines (4) through (6), (7) through (10), and (14) through (18); it is false for the element of line (11). □

```
<Products>
    <Maker name = "A">
        <PC model = "1001" price = "2114">
            <Speed>2.66</Speed>
            <RAM>1024</RAM>
            <HardDisk>250</HardDisk>
        </PC>
        <PC model = "1002" price = "995">
            <Speed>2.10</Speed>
            <RAM>512</RAM>
            <HardDisk>250</HardDisk>
        </PC>
        <Laptop model = "2004" price = "1150">
            <Speed>2.00</Speed>
            <RAM>512</RAM>
            <HardDisk>60</HardDisk>
            <Screen>13.3</Screen>
        </Laptop>
        <Laptop model = "2005" price = "2500">
            <Speed>2.16</Speed>
            <RAM>1024</RAM>
            <HardDisk>120</HardDisk>
            <Screen>17.0</Screen>
        </Laptop>
    </Maker>
```

Figure 12.4: XML document with product data — beginning

12.1.10 Exercises for Section 12.1

Exercise 12.1.1: Figures 12.4 and 12.5 are the beginning and end, respectively, of an XML document that contains some of the data from our running products exercise. Write the following XPath queries. What is the result of each?

```
    <Maker name = "E">
        <PC model = "1011" price = "959">
            <Speed>1.86</Speed>
            <RAM>2048</RAM>
            <HardDisk>160</HardDisk>
        </PC>
        <PC model = "1012" price = "649">
            <Speed>2.80</Speed>
            <RAM>1024</RAM>
            <HardDisk>160</HardDisk>
        </PC>
        <Laptop model = "2001" price = "3673">
            <Speed>2.00</Speed>
            <RAM>2048</RAM>
            <HardDisk>240</HardDisk>
            <Screen>20.1</Screen>
        </Laptop>
        <Printer model = "3002" price = "239">
            <Color>false</Color>
            <Type>laser</Type>
        </Printer>
    <Maker name = "H">
        <Printer model = "3006" price = "100">
            <Color>true</Color>
            <Type>ink-jet</Type>
        </Printer>
        <Printer model = "3007" price = "200">
            <Color>true</Color>
            <Type>laser</Type>
        </Printer>
    </Maker>
</Products>
```

Figure 12.5: XML document with product data — end

a) Find the model numbers of PC's with a hard disk of at least 300 gigabytes.

b) Find the amount of hard disk on each PC.

c) Find the price of each product of any kind.

d) Find all the laptop elements.

! e) Find the makers of color printers.

! f) Find the makers of PC's and/or laptops.

!! g) Find the makers of at least two printers.

Exercise 12.1.2: The document of Fig. 12.6 contains data similar to that used in our running battleships exercise. In this document, data about ships is nested within their class element, and information about battles appears inside each ship element. Write the following queries in XPath. What is the result of each?

a) Find the names of all ships.

b) Find the names of the ships that were damaged.

c) Find all the `Class` elements for classes with a displacement larger than 40000.

d) Find all the `Ship` elements for ships that were launched before 1927.

! e) Find the names of all ships that were in battles.

! f) Find the years in which ships having the same name as their class were launched.

!! g) Find the `Ship` elements for all ships that fought in three or more battles.

12.2 XQuery

XQuery is an extension of XPath that has become a standard for high-level querying of databases containing data in XML form. This section will introduce some of the important capabilities of XQuery.


```
<Ships>
    <Class nam
            nu
        <Ship
        <Ship
        <Ship
            <B
        </Ship
        <Ship
    </Class>
    <Class nam
            nu
        <Ship
        <Ship
            <B
        </Ship
    </Class>
    <Class nam
            nu
        <Ship
            <B
        </Ship
        <Ship
            <B
    </Class>
    <Class nam
            co
            nu
        <Ship
        <Ship
            <B
            <B
        </Ship
        <Ship
            <B
        </Ship
        <Ship
        <Ship
    </Class>
</Ships>
```

Figur

to break ties alphabetically by title.

After sorting the bindings, each binding is passed to the return-clause, in the order chosen. By substituting for the variables in the return-clause, we produce from each binding a single `Movie` element. □

12.2.11 Exercises for Section 12.2

Exercise 12.2.1: Using the product data from Figs. 12.4 and 12.5, write the following in XQuery.

a) Find the `Laptop` elements with a price less than 800.

b) Find the `Laptop` elements with a price less than 800, and produce the sequence of these elements surrounded by a tag `<CheapLaptops>`.

! c) Find the makers such that every laptop they produce has a price no more than 1000.

! d) Find the names of the makers of both PC's and laptops.

! e) Find the names of the makers that produce at least two PC's with a speed of 2.80 or more.

!! f) Produce a sequence of elements of the form

`<PC><Model>`x`</Model><Maker>`y`</Maker></PC>`

where x is the model number and y is the name of the maker of the PC.

Exercise 12.2.2: Using the battleships data of Fig. 12.6, write the following in XQuery.

a) Find the names of the ships that had at least 12 guns.

b) Find the names of the ships that were damaged.

c) Find the names of the classes with at least 4 ships.

d) Find the names of the classes with a displacement at least 40,000.

! e) Find the names of the classes such that no ship of that class was in a battle.

!! f) Find the names of the classes that had at least two ships launched in the same year.

!! g) Produce a sequence of items of the form

`<Battle name =` x`><Ship name =` y `/>···</Battle>`

where x is the name of a battle and y the name of a ship in the battle. There may be more than one `Ship` element in the sequence.

! **Exercise 12.2.3:** Do there exist expressions E and F such that the expression `every $x in` E `satisfies` F is true, but `some $x in` E `satisfies` F is false? Either give an example or explain why it is impossible.

! **Exercise 12.2.4:** Solve the problem of Section 12.2.5; write a query that finds the star(s) living at a given address, even if they have several addresses, without finding stars that do not live at that address.

12.3 Extensible Stylesheet Language

XSLT (Extensible Stylesheet Language for Transformations) is a standard of the World-Wide-Web Consortium. Its original purpose was to allow XML documents to be transformed into HTML or similar forms that allowed the document to be viewed or printed. However, in practice, XSLT is another query language for XML. Like XPath or XQuery, we can use XSLT to extract data from documents or turn one document form into another form.

12.3.1 XSLT Basics

Like XML Schema, XSLT specifications are XML documents; these specifications are usually called *stylesheets*. The tags used in XSLT are found in a namespace, which is `http://www.w3.org/1999/XSL/Transform`. Thus, at the highest level, a stylesheet looks like Fig. 12.20.

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

Figure 12.20: The form of an XSLT stylesheet

12.3.2 Templates

A stylesheet will have one or more *templates*. To apply a stylesheet to an XML document, we go down the list of templates until we find one that matches the root. As processing proceeds, we often need to find matching templates for elements nested within the document. If so, we again search the list of templates for a match according to matching rules that we shall learn in this section. The simplest form of a template tag is:

12.3.6 Conditionals in XSLT

We can introduce branching into our templates by using an `if` tag. The form of this tag is:

```
<xsl:if test = "boolean expression">
```

Whatever appears between this tag and its matched closing tag is executed if and only if the boolean expression is true. There is no else-clause, but we can follow this expression by another `if` that has the opposite test condition should we wish.

```
 1)  <? xml version = "1.0" encoding = "utf-8" ?>
 2)  <xsl:stylesheet xmlns:xsl =
 3)          "http://www.w3.org/1999/XSL/Transform">
 4)      <xsl:template match = "/">
 5)          <TABLE border = "5"><TR><TH>Stars</th></tr>
 6)          <xsl:for-each select = "Stars/Star" />
 7)              <xsl:if test = "Address/City = 'Hollywood'">
 8)                  <TR><TD>
 9)                  <xsl:value-of select = "Name" />
10)                  </td></tr>
11)              </xsl:if>
12)          </xsl:for-each>
13)          </table>
14)      </xsl:template>
15)  </xsl:stylesheet>
```

Figure 12.28: Finding the names of the stars who live in Hollywood

Example 12.24: Figure 12.28 is a stylesheet that prints a one-column table, with header "Stars." There is one template, which matches the root. The first thing this template does is print the header row at line (5). The for-each loop of lines (6) through (12) iterates over each star. The conditional of line (7) tests whether the star has at least one home in Hollywood. Remember that the equal-sign represents a comparison is true if any item on the left equals any item on the right. That is what we want, since we asked whether any of the homes a star has is in Hollywood. Lines (8) through (10) print a row of the table. □

12.3.7 Exercises for Section 12.3

Exercise 12.3.1: Suppose our input XML document has the form of the product data of Figs. 12.4 and 12.5. Write XSLT stylesheets to produce each of the following documents.

a) An HTML file consisting of a table with headers "Model" and "Price," with a row for each laptop. That row should have the proper model and price for the laptop.

! b) Repeat part (a), but make the output file a Latex file.

c) An HTML file consisting of a header "Manufacturers" followed by an enumerated list of the names of all the makers of products listed in the input.

! d) An HTML file consisting of a table whose headers are "Model," "Price," "Speed," and "Ram" for all PC's, followed by another table with the same headers for laptops.

e) An XML file with root tag `<PCs>` and subelements having tag `<PC>`. This tag has attributes `model`, `price`, `speed`, and `ram`. In the output, there should be one `<PC>` element for each `<PC>` element of the input file, and the values of the attributes should be taken from the corresponding input element.

!! f) An XML file with root tag `<Products>` whose subelements are `<Product>` elements. Each `<Product>` element has attributes `type`, `maker`, `model`, and `price`, where the type is one of `"PC"`, `"Laptop"`, or `"Printer"`. There should be one `<Product>` element in the output for every PC, laptop, and printer in the input file, and the output values should be chosen appropriately from the input data.

Exercise 12.3.2: Suppose our input XML document has the form of the product data of Fig. 12.6. Write XSLT stylesheets to produce each of the following documents.

a) An XML file identical to the input, except that `<Battle>` elements should be empty, with the outcome and name of the battle as two attributes.

b) An HTML file with a header for each class. Under each header is a table with column-headers "Name" and "Launched" with the appropriate entry for each ship of the class.

c) An HTML file with root tag `<Losers>` and subelements `<Ship>`, each of whose values is the name of one of the ships that were sunk.

! d) An XML file with root tag `<Ships>` and subelements `<Ship>` for each ship. These elements each should have attributes `name`, `class`, `country` and `displacement` with the appropriate values taken from the input file.

! e) Repeat (d), but only list those ships that were in at least one battle.