BT5110 Week 9 Lecture
Introduction to XML

Chan Chee Yong
chancy@comp.nus.edu.sg

Learning Objectives

- **What you've learned so far**:
  - How to import spreadsheet data (in CSV format) into relational tables
  - How to query relational data using SQL
- **What you'll learn today**: How to manipulate textual data
  - How to represent data using XML & DTD
  - How to extract/transform textual XML/HTML data using XPath and XSLT
- Important for data analysts to acquire a repertoire of data manipulation skills

XML

- Simplified subset of SGML for data representation
- Features:
  - extensible
  - self-describing
  - machine & human readable
  - platform & vendor independent
- Useful for data exchange & data integration
- XML 1.0 – W3C Recommendation (February 1998)
- XML 1.0 (5^{th} edition) – W3C Recommendation (November 2008)

XML: Example

```
<? xml version="1.0"   encoding="UTF-8"   standalone="yes" ? >
<books>
    <book     price="30"     year="1978" >
        <author>Dennis Ritchie</author>
        <author>Brian Kernighan</author>
        <title>C Programming Language</title>
    </book>
    <book    price="15"     currency="USD">
        <title>2001: Space Odyssey</title>
        <author>
            <firstname>Arthur</firstname>
            <lastname>Clarke</lastname>
        </author>
    </book>
</books>
```

- XML data consists of nested elements
- Each element is represented by a pair of **opening** & **closing tags**
- Elements can have attributes & values

XML data is everywhere!

- Microsoft's Word, Excel, Powerpoint files
- Scalable Vector Graphics (SVG) files
- GPS Exchange Format (GPX) files
- OpenStreetMap (OSM) files
- Keyhole Markup Language (KML) files
- etc.

Well-Formed XML

- Begins with a **declaration** that it is XML
- Contains exactly one **root element**
- Tags are properly nested
- Any attributes associated with each element are uniquely named

```
<? xml version="1.0"   encoding="UTF-8"   standalone="yes" ? >
<books>
    <book     price="30"     year="1978" >
        <author>Dennis Ritchie</author>
        <author>Brian Kernighan</author>
        <title>C Programming Language</title>
    </book>
    <book    price="15"     currency="USD">
        <title>2001: Space Odyssey</title>
        <author>
            <firstname>Arthur</firstname>
            <lastname>Clarke</lastname>
        </author>
    </book>
</books>
```

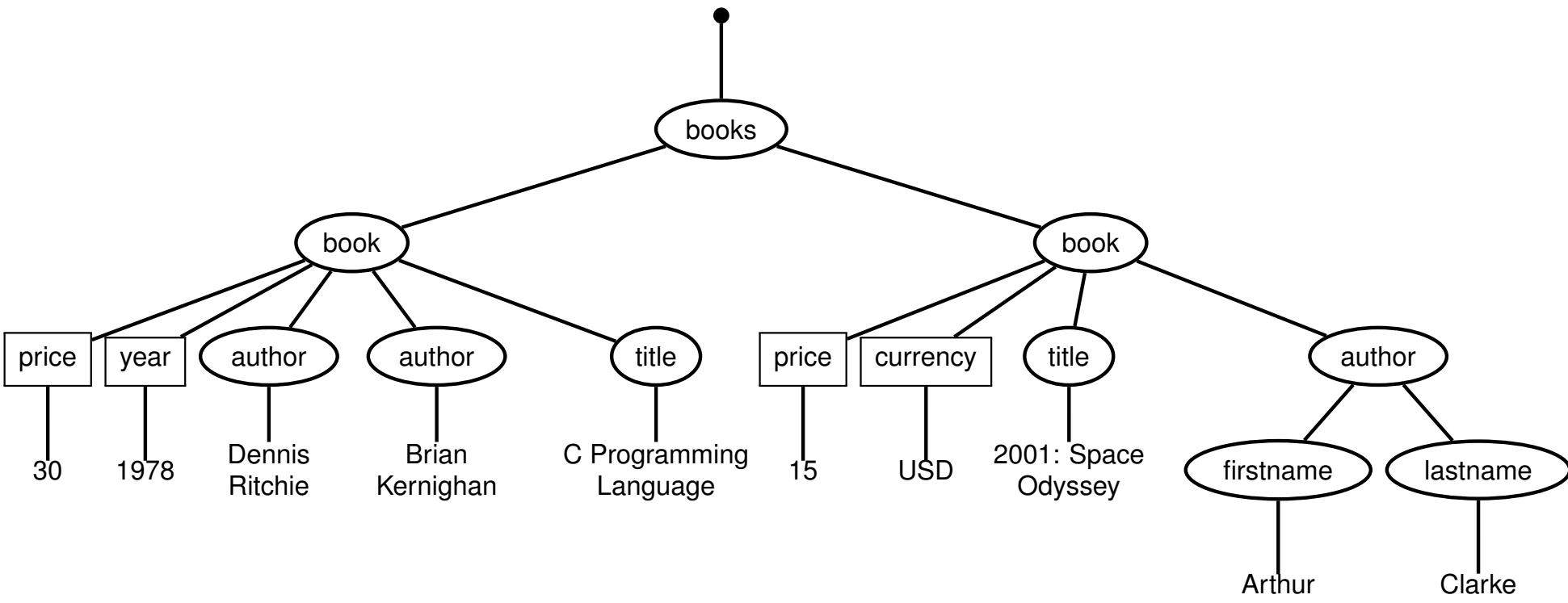
Well-Formed XML (cont.)

- The following XML document is not well-formed

```
<book year="1978" >
        <author>Dennis Ritchie</author>
        <author>Brian Kernighan</author>
        <title>C Programming Language</title>
</book>
<book    price="15"    currency="USD">
        <title>2001: Space Odyssey</title>
        <author>
            <firstname>Arthur</firstname>
            <lastname>Clarke</author>
        </lastname>
</book>
```

XML: Tree Representation

- XML document is modeled as a tree of nodes
- Seven types of nodes: document (root), element, attribute, text, comment, processing instruction, namespace
- Tree nodes are totally ordered based on document order



XML: Node Types

- **Element**
- **Attribute**
- **Text**
- **Comment**
- **Processing instruction**
- **Document (root)**
  - Every XML tree starts with a single document/root node.
  - Children of root node:
    - Any number of comment or processing instruction nodes
    - Exactly one root element node
    - Root element not to be confused with root node

XML: Processing Instructions

- Processing instructions are of the form: <? instruction ?>
- Example of **XML declaration**:

<? xml version="1.0" encoding="UTF-8" ? >

XML: Comments

- Comments can be specified in one of the following tag forms:
  - <- - this is a comment - ->
  - <- this is a comment ->
  - <!- - this is a comment - ->
- Comments can't appear before the XML declaration
- Comments can't appear inside element tags

XML: Elements

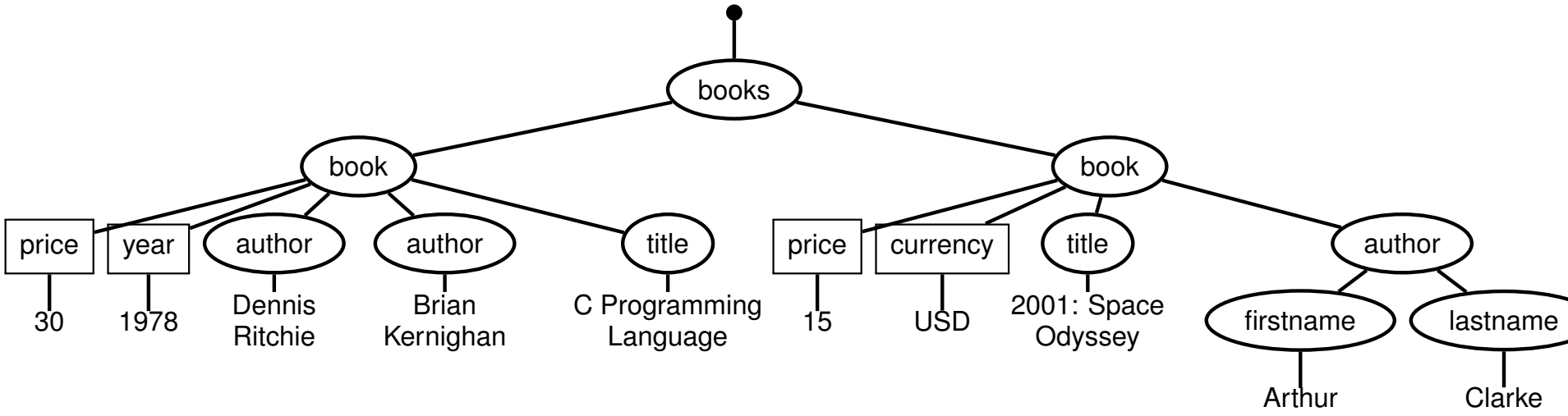
- Element names cannot contain a space character and any of of these characters:

  `!"\#$%&'()*+,/;<=>?@[\]^‘{|}~`

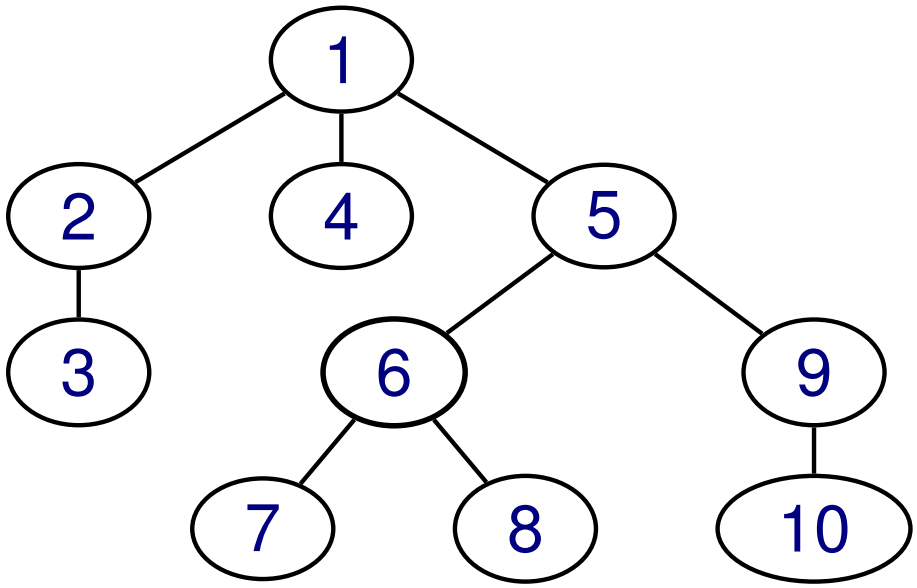
- Element names cannot start with a numeric digit, ., -, or the three letters “xml” (in lower, upper or mixed case)
- Each element has an **opening** and **closing tag**
  - Non-empty element: <book> ... </book>
  - Empty element: <book/>
- An non-empty element’s opening tag (or empty element’s tag) may be followed by a list of **attributes** and their **values**
  - <book year="1972" publisher="Penguin" >
- An element name may be preceded by a **name space** name separated by a colon
  - <Penguin:book>

XML: Document Order

```
<? xml version="1.0"   encoding="UTF-8"   standalone="yes" ? >
<books>
      <book      price="30"      year="1978" >
            <author>Dennis Ritchie</author>
            <author>Brian Kernighan</author>
            <title>C Programming Language</title>
      </book>
      <book      price="15"      currency="USD">
            <title>2001: Space Odyssey</title>
            <author>
                  <firstname>Arthur</firstname>
                  <lastname>Clarke</lastname>
            </author>
      </book>
</books>
```



XML: Document Order (cont.)



- Document order = Pre-order traversal of document tree
- A node X precedes Y (or Y follows X) in document order if X occurs before Y in document

XML: Schema Languages

- XML data is schema-less, but could include a schema specification
- Two popular schema languages:
  - Document Type Definition (DTD)
  - XML Schema Definition Language (XSD)
- Grammar for describing document structure & constraints
- An XML document is valid w.r.t. a schema if the document conforms to the schema

DTD: Example

```
<? xml version="1.0" standalone="yes" ? >
<!- - Example DTD for books - ->
<!DOCTYPE books [
<!ELEMENT   books      (book*) >
<!ELEMENT   book       ( author+, title, price, review?) >
<!ATTLIST   book       isbn  ID  #REQUIRED >
<!ATTLIST   book       year  CDATA  #REQUIRED >
<!ATTLIST   book       publisher  CDATA  >
<!ELEMENT   author     (name | (firstname, lastname)) >
<!ELEMENT   firstname  (#PCDATA) >
<!ELEMENT   lastname   (#PCDATA) >
<!ELEMENT   name       (#PCDATA) >
<!ELEMENT   price      (#PCDATA) >
<!ELEMENT   title      (#PCDATA) >
<!ELEMENT   review     (#PCDATA) >
] >
```

books.dtd

DTD: Example (cont.)

```
<? xml version="1.0" encoding="utf-8" standalone="no" ? >
<!DOCTYPE Books SYSTEM "books.dtd" >
<books>
    <book isbn = "12345678" year = "1865" >
        <author>
            <name> Lewis Carroll </name>
        </author>
        <title> Alice in Wonderland </title>
        <price> 25 </price>
    <book>
    <book isbn = "1000000" year = "1960" publisher = "L.B.L." >
        <author>
            <firstname> Harper </firstname>
            <lastname> Lee </lastname>
        </author>
        <title> To Kill a Mockingbird</title>
        <price> 20 </price>
        <review> Deeply moving story </review>
    <book>
</books>
```

books.xml

DTD: Example (cont.)

- Root element is **books**
- Each **books** element contains zero or more **book** sub-elements
- Each **book** element contains the following sequence of attributes:
  - **isbn** which is mandatory
  - **year** which is mandatory
  - **publisher** which is optional
- Each **book** element contains the following sequence of sub-elements:
  - One or more **author**
  - One **title**
  - One **price**
  - At most one **review** (possibly none)

DTD: Example (cont.)

- Each **author** element contains the following sub-elements:
  - Either only **name** element, or
  - **firstname** followed by **lastname** elements
- Data types
  - **PCDATA** = Parsed Character data
    - Any characters are allowed except the following: <, &,]] >
  - **CDATA** = Character data
    - Similar restrictions as PCDATA
  - **ID** = identifier data
    - Value serves as unique identifier for element
    - Can be referenced from other elements using attributes of type **IDREF** or **IDREFS**

Entity References

- Entity references are used as substitutions for specific characters in XML

Entity	Character
<lt;	<
<gt;	>
<amp;	&
<apos;	’
<quot;	"

Querying XML

- Major languages:
  - XPath
  - XSLT
  - XQuery
  - SQL/XML

XPath Language

- Language for selecting nodes in XML documents
  - Version 1.0 – W3C recommendation (November 1999)
  - Version 2.0 – W3C recommendation (December 2010)
  - Version 3.0 – W3C recommendation (April 2014)
  - Version 3.1 – W3C recommendation (March 2017)
- Used as a sub-language in XSLT, XQuery, XML Schema, etc.
- Focus here is on XPath 1.0

XPath: Key Concepts

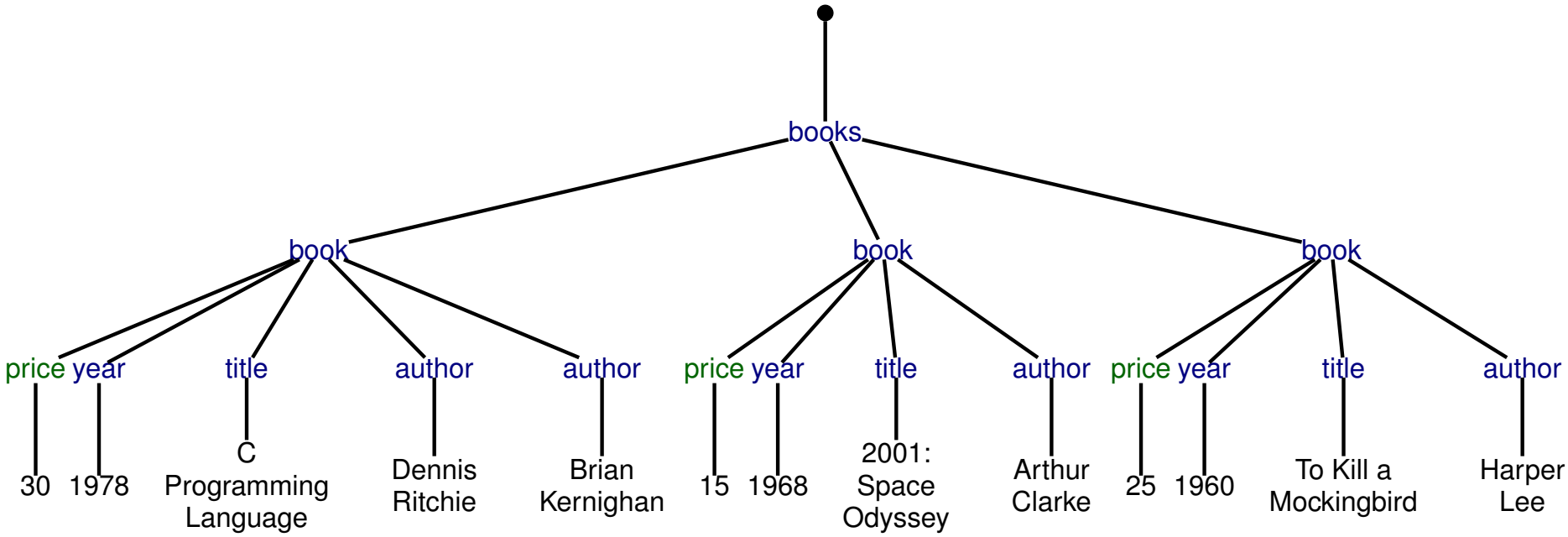
- Each XPath query specifies a **path expression**
- Path expression = sequence of **location steps**

$$step_1 \ / \ step_2 \ / \ \cdots \ / \ step_n$$

- Each location step is evaluated w.r.t. a **context node**
  - Context node for first step = document's root node
- Each step's evaluation returns a set of nodes / values
- Each node returned by the previous step is used as a context node for next step's evaluation

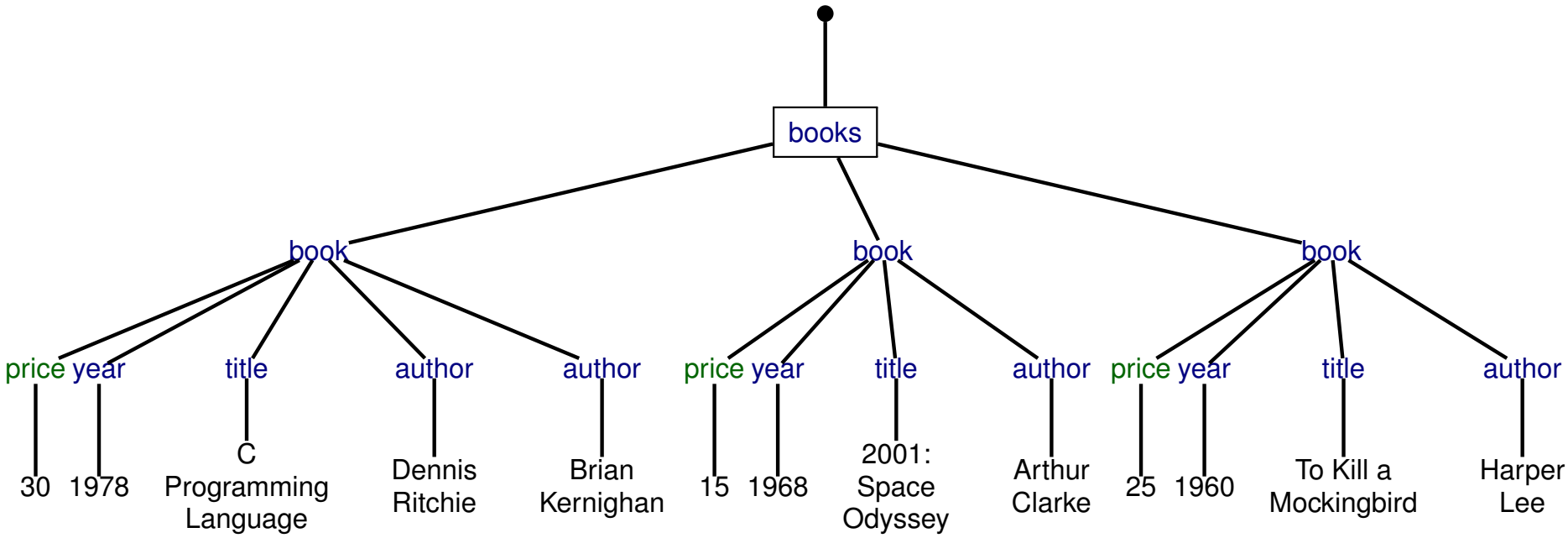
Example XML Document Tree

Assume that price is the only attribute in XML document

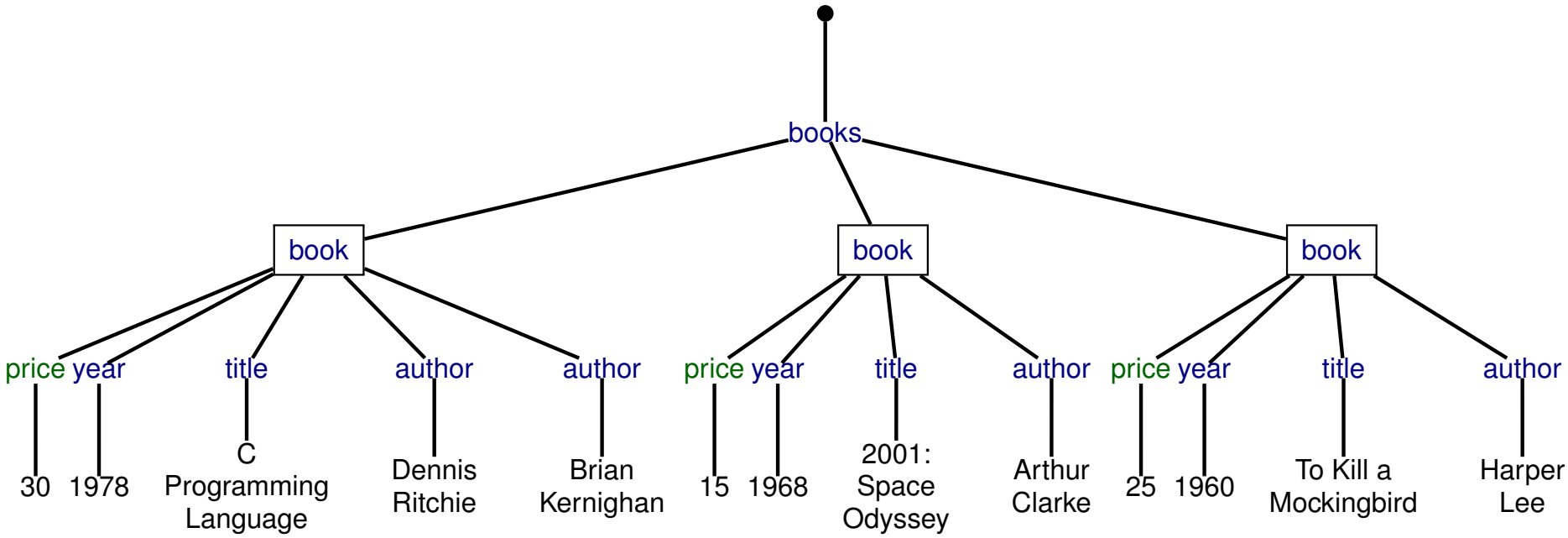


Example 1: /books

/ represents a parent-child relationship
/books find all child elements of context node named "books"

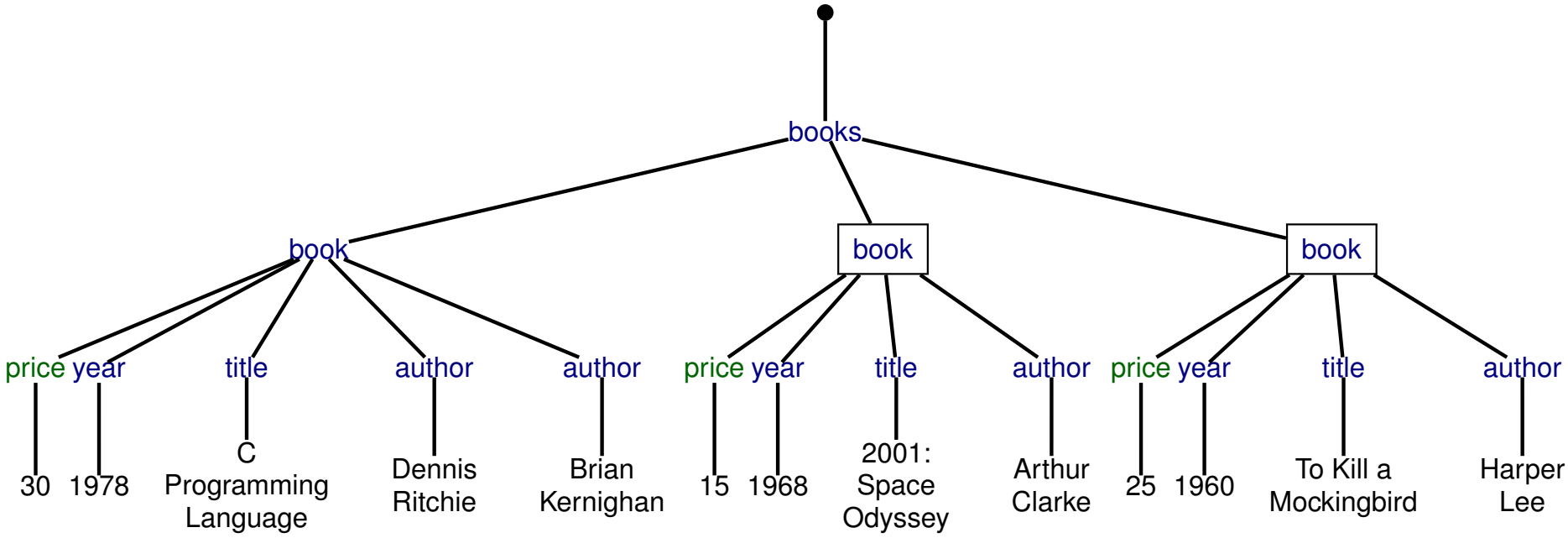


Example 2: /books/book



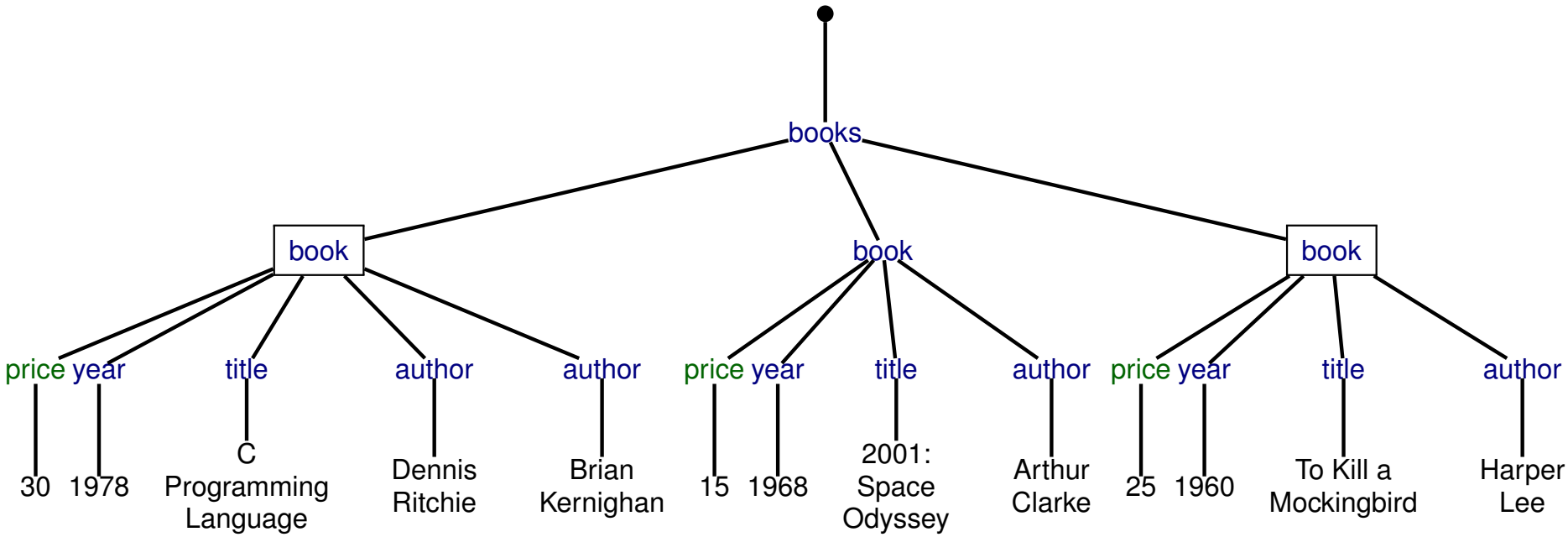
Example 3: /books/book[year $\leq$ 1970]

[predicate] specifies a condition that must be satisfied by matching element nodes

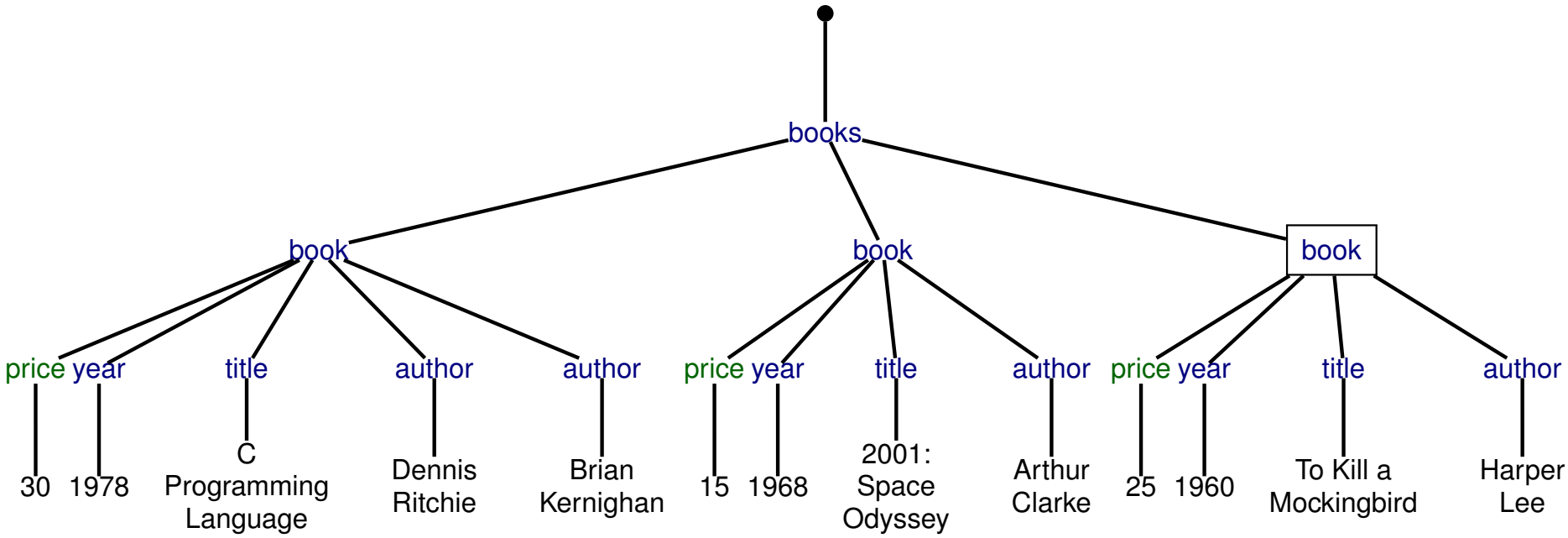


Example 4: /books/book[@price > 20]

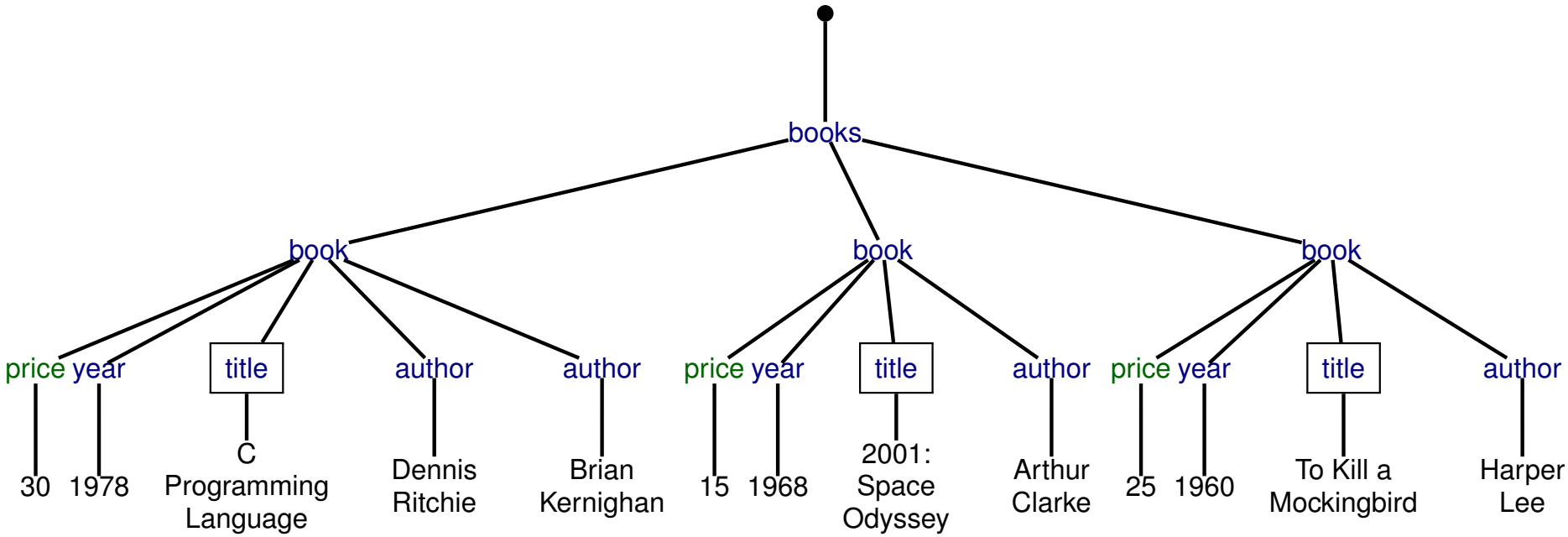
@attribute refers to an attribute of matching element nodes



Example 5: /books/book[@price $>$ 20][year $\leq$ 1970]



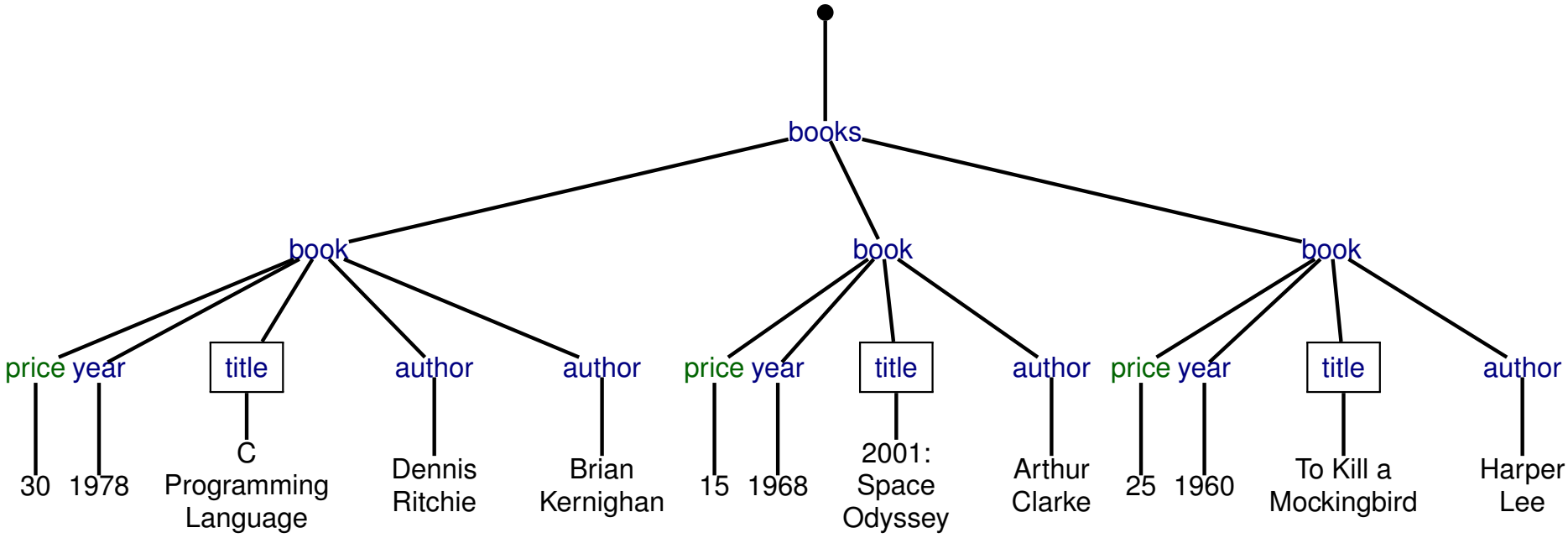
Example 6: /books/book/title



Example 7: //title

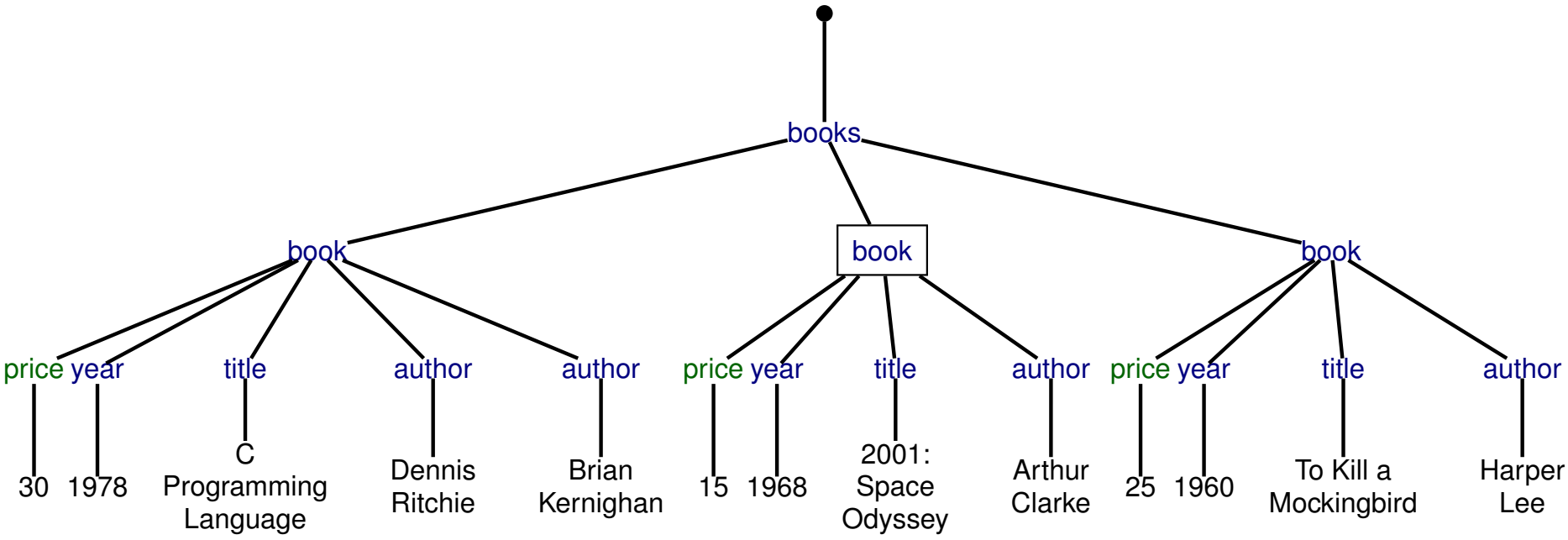
// represents an ancestor-descendant relationship

//title finds descendant elements of context node named "title"



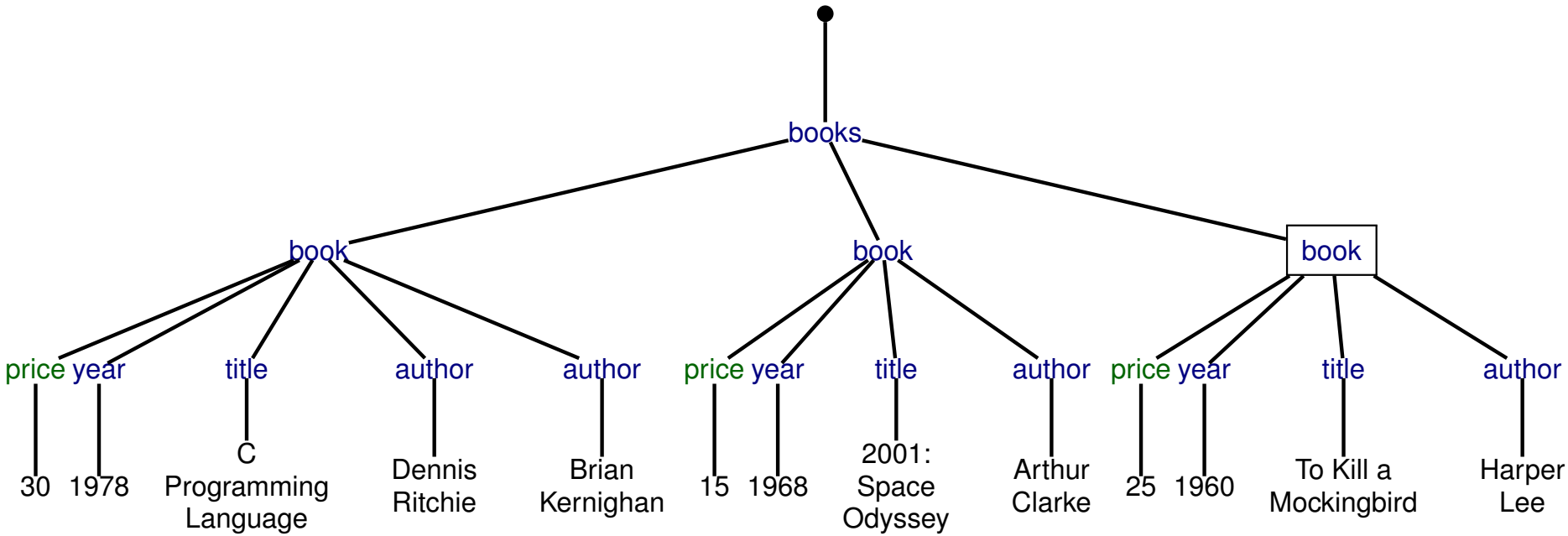
Example 8: //book[2]

[n] is an abbreviation for [position() = n] which selects the n^{th} matching element



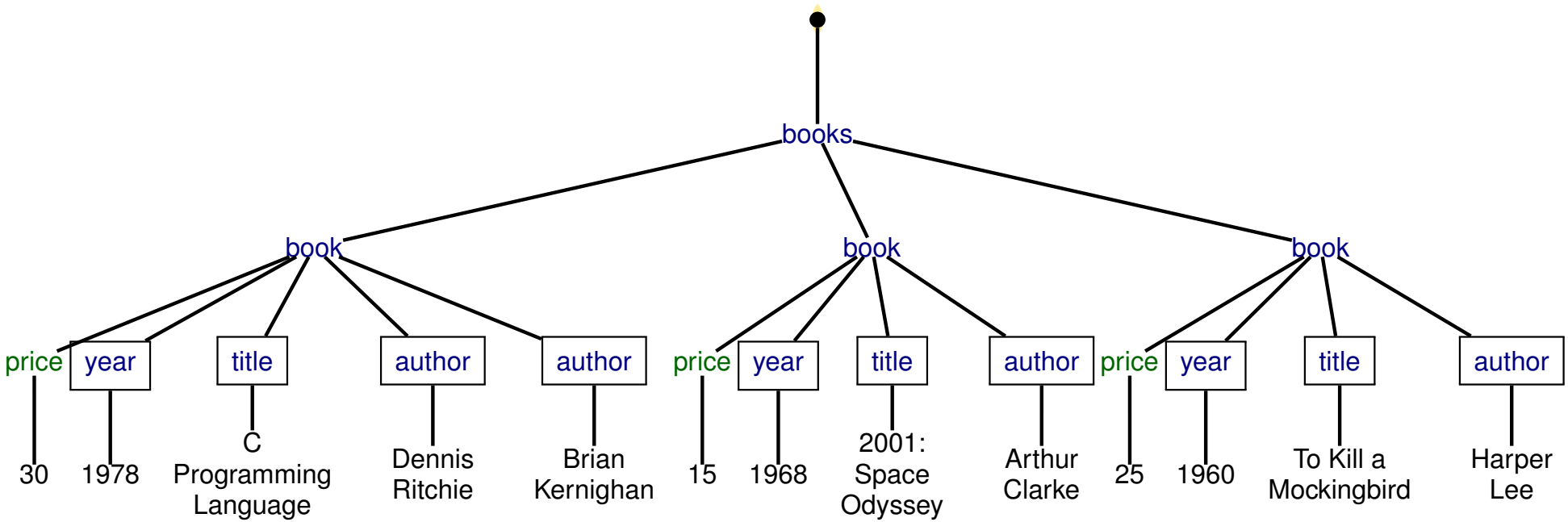
Example 9: //book[last()]

[last()] selects the last matching element

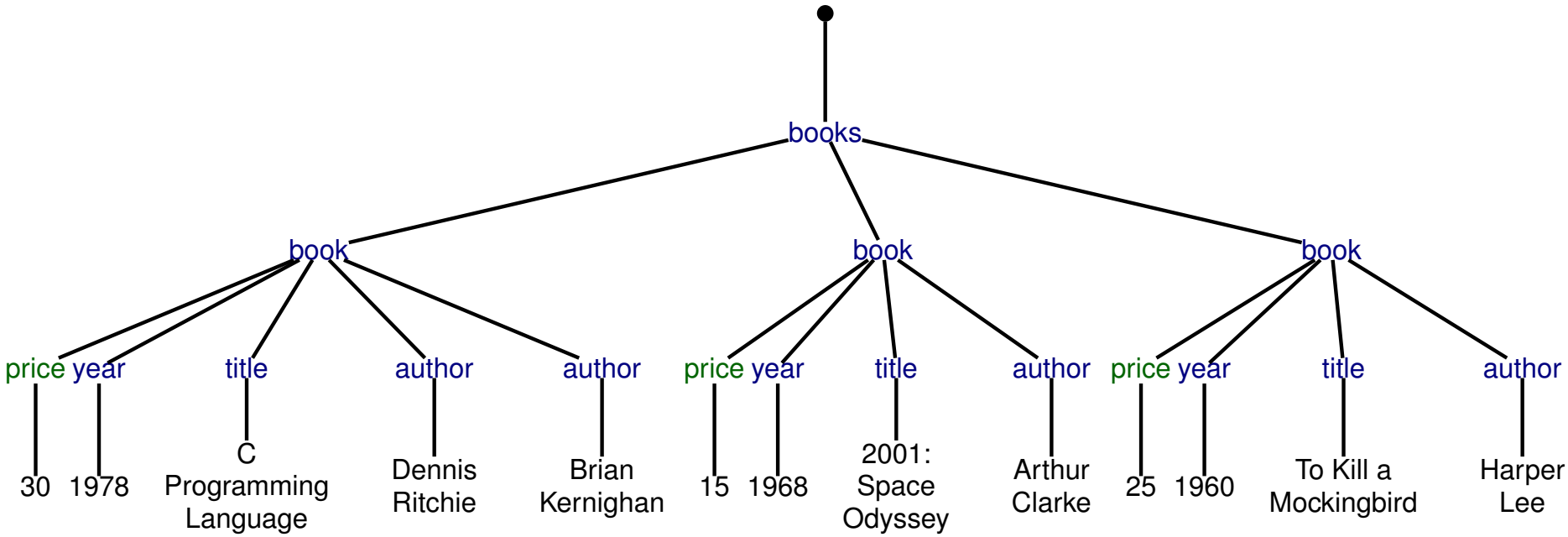


Example 10: //book/*

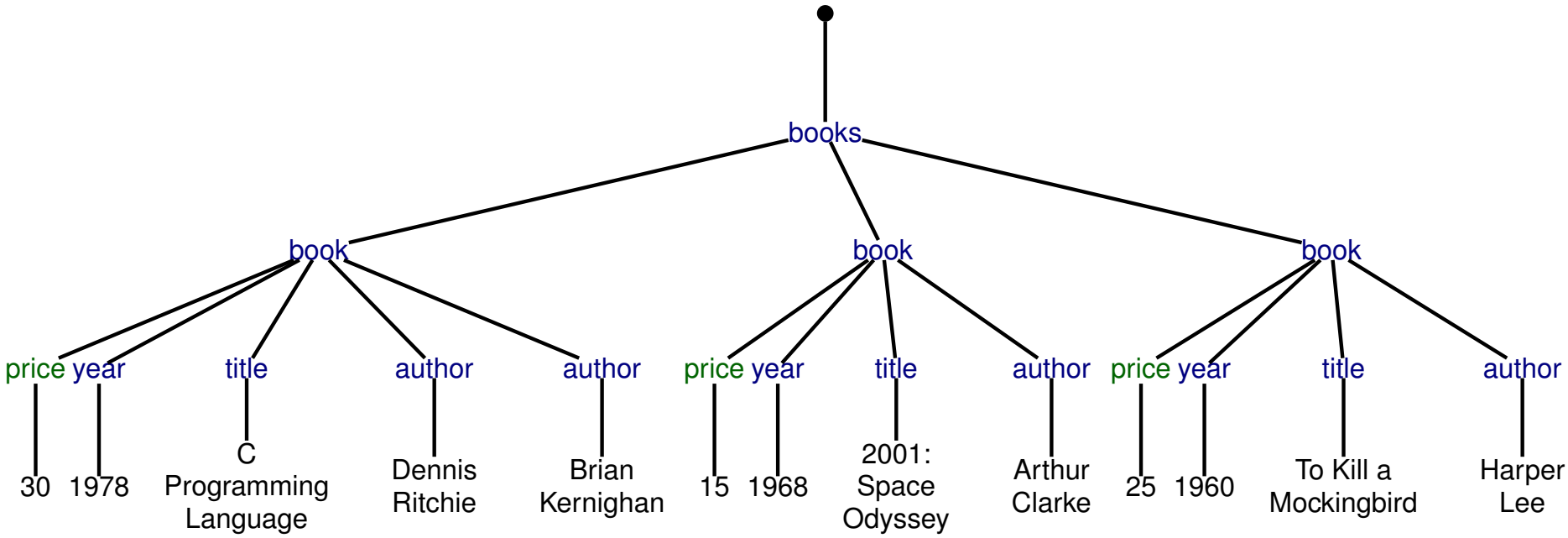
* represents any element



Example 11: What does /*/* return?

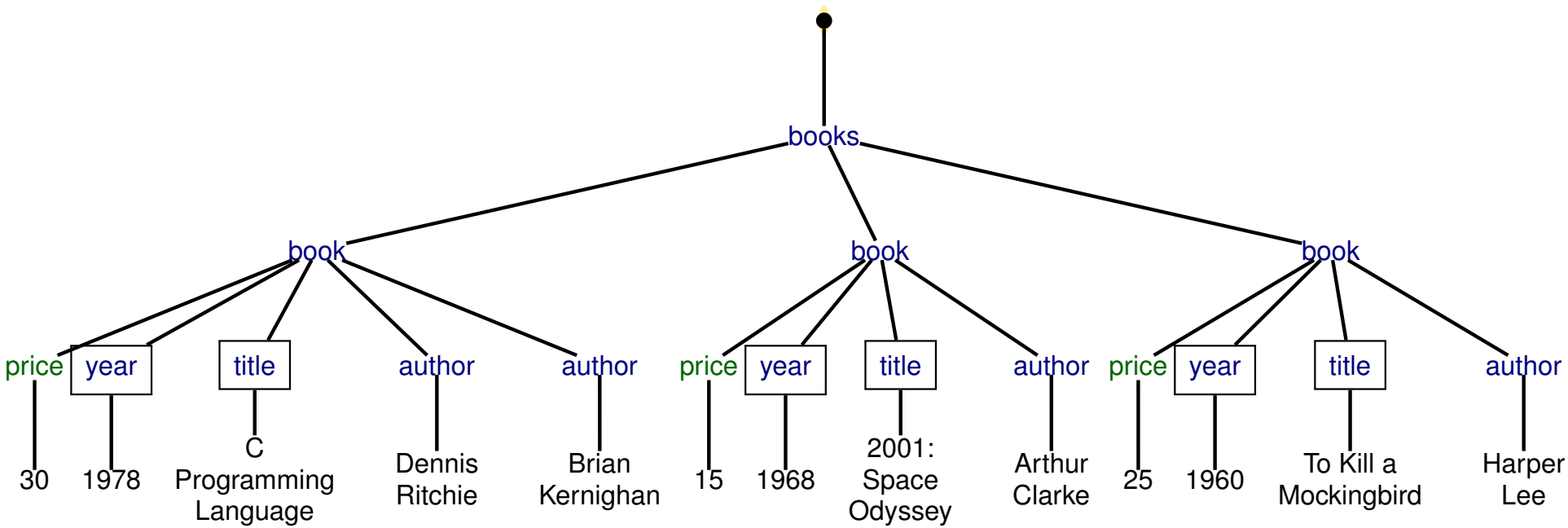


Example 12: What does //*[year $\leq$ 1970]/* return?



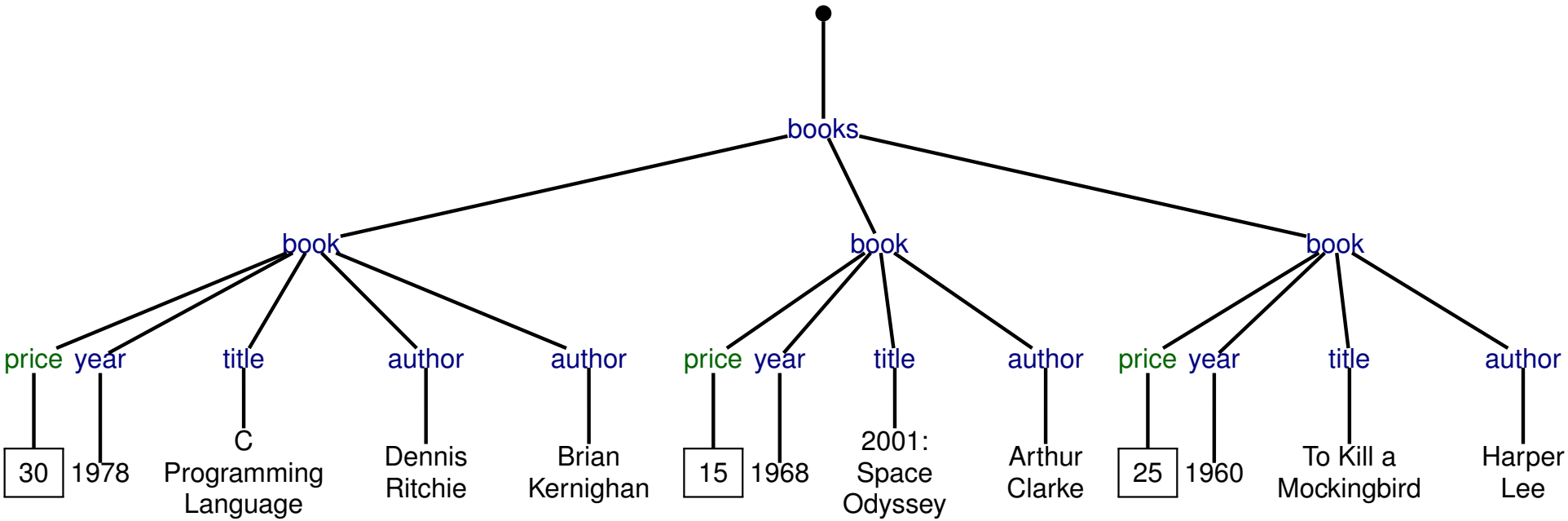
Example 13: (//title | //year)

| represents union



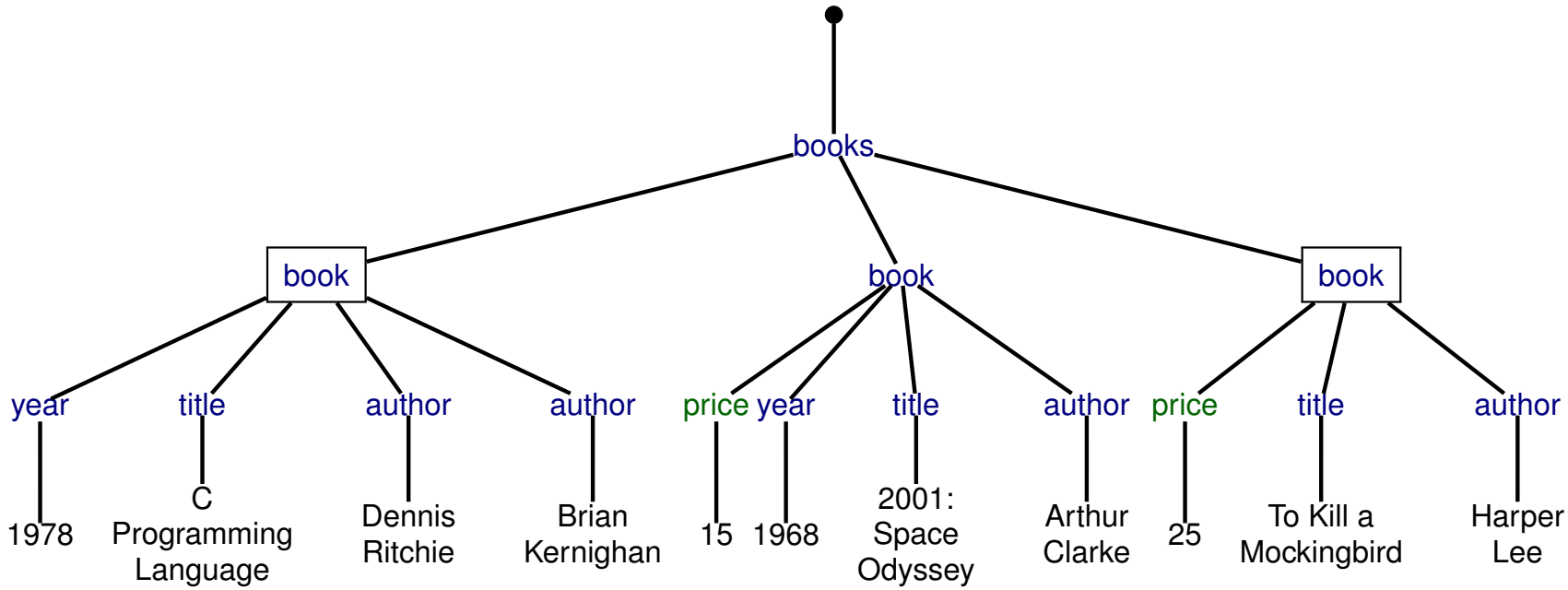
Example 14: /books/book/data(@price)

data(X) returns the value of *X*

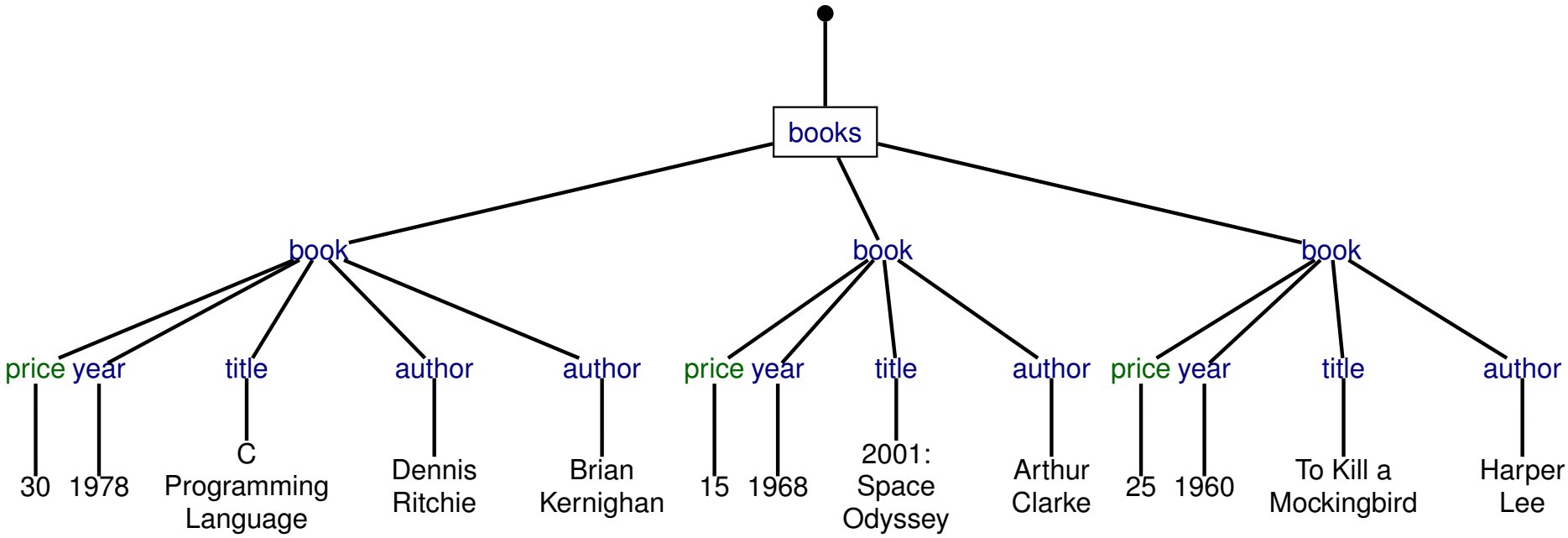


Example 15: //book[(not @price) or (not year)]

要不没有price 要不没有year的book

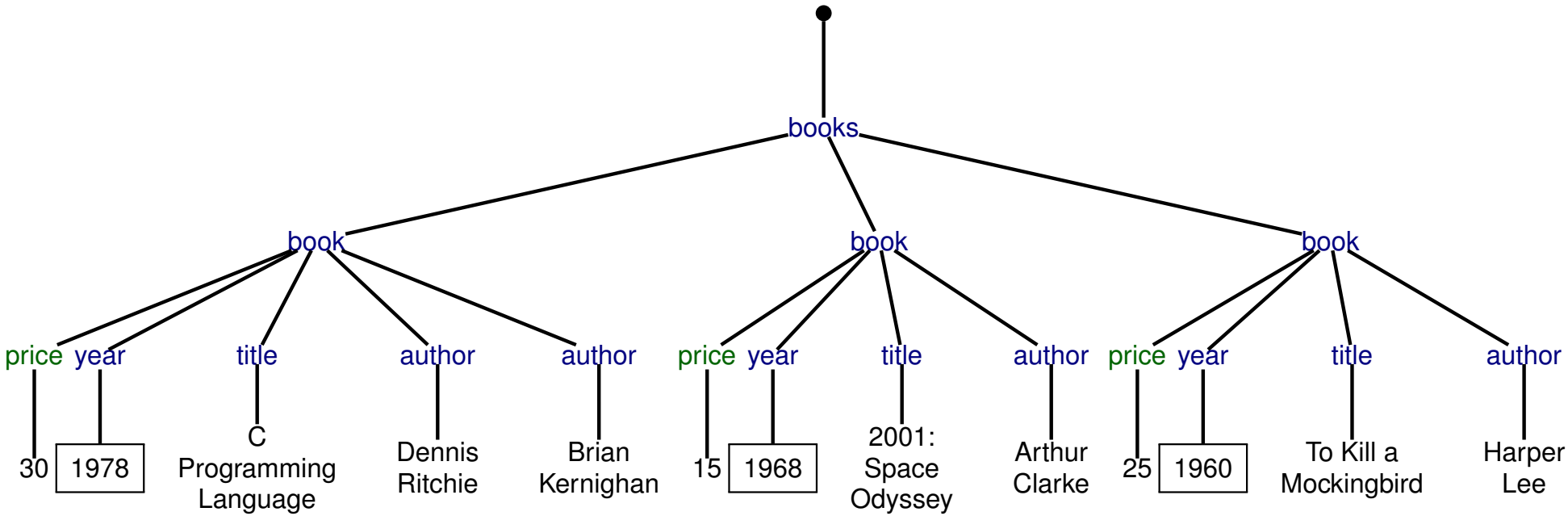


Example 16: /books[book/year = 1978]



Example 17: //year/text()

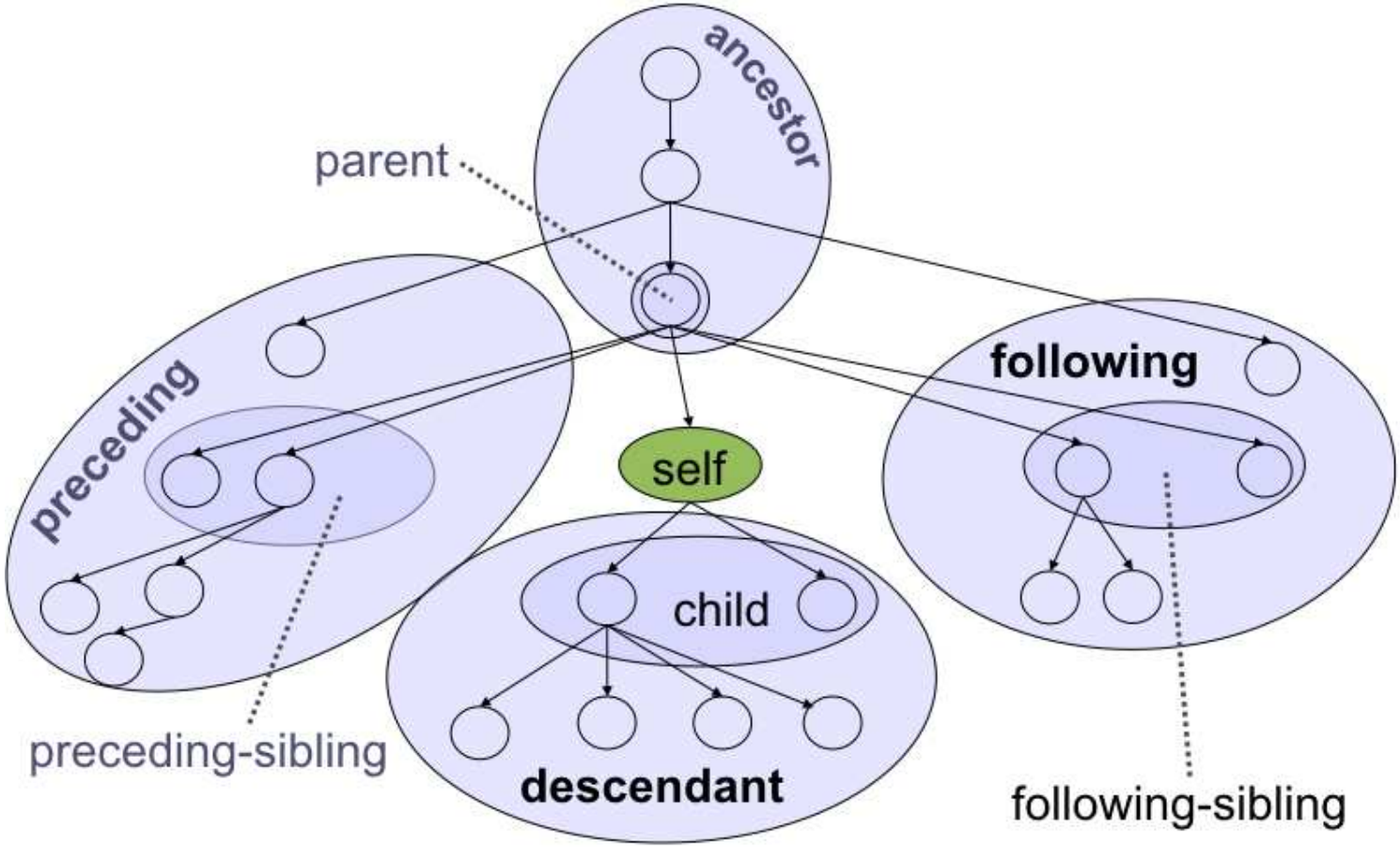
text() finds child text node of context node



XPath: Location Steps

- **Location step** is of the form: axis::node test predicate*
- Axis specifies relationship between context & selected nodes
  - self
  - **Forward axes**: child, descendant, following, following-sibling,
  - **Backward axes**: parent, ancestor, preceding, preceding-sibling
  - ancestor-or-self
  - descendant-or-self
  - attribute

XPath Axes



(Source: Benjamin Piworwarski)

XPath Axes

- Let c denote a context node
- **self**: c itself
- **child**: c's children
- **descendant**: c's children, c's grandchilden, ...
- **parent**: c's parent (empty if c is root element)
- **ancestor**: c's parent, c's grandparent, ...
- **following-sibling**: siblings of c that follow c (in document order)
- **preceding-sibling**: siblings of c that precede c (in document order)
- **following**: all nodes following c excluding c's descendants
- **preceding**: all nodes preceding c excluding c's ancestors
- **descendant-or-self**: union of descendant and self
- **ancestor-or-self**: union of ancestor and self

XPath: Node Tests

- **Location step** is of the form: axis::node test predicate*
- Node test specifies the type of selected nodes
  - element name selects specified element nodes
  - * selects all element nodes
  - attribute name selects specified attribute nodes
  - text() selects text nodes
  - comment() selects comment nodes
  - processing-instruction() selects processing-instruction nodes
  - node() selects nodes of any type excluding attributes & namespace declarations

XPath: Predicates

- **Location step** is of the form: axis::nodeTest predicate*
- Predicates further refine selected nodes
  - Predicates are optional
  - Each predicate is an expression enclosed in square brackets

- **Example:** *Return title of books published in 2003 that cost less than $10.*

  /descendant::book [child::year = 2003] [attribute::price < 10] /
  child::title

XPath Abbreviations

Abbreviation	Meaning
.	self::node()
..	parent::*
/books	/child::books
/node()	/child::node()
/*	/child::*
//	/descendant-or-self::node()
*	all elements
@*	all attributes

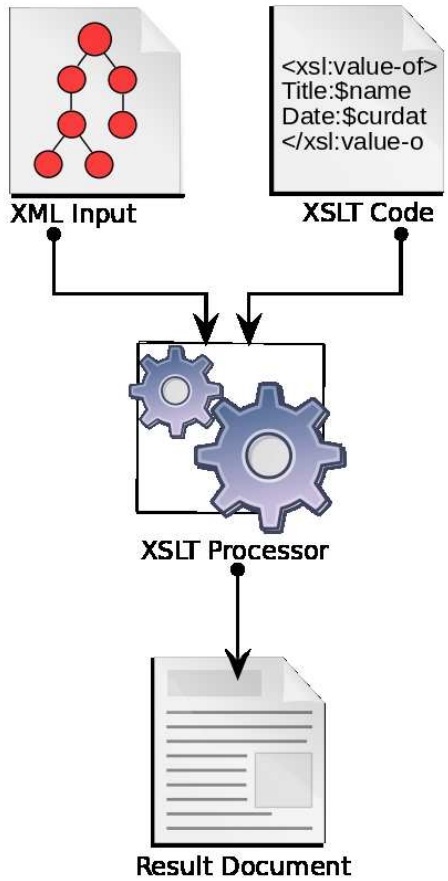
XPath: Other Expressions

- Function expressions: position(), first(), last(), etc.
- Union expressions: expr1 | expr2 | $\cdots$ | exprN
- Arithmetic expressions
- Boolean expressions
- etc.

XSLT

- XSLT = Extensible Stylesheet Language for Transformations
- Language for transforming XML documents
  - Version 1.0 – W3C recommendation (November 1999)
  - Version 2.0 – W3C recommendation (June 2007)
  - Version 3.0 – W3C recommendation (June 2017)

XSLT Processing



(Source: Wikipedia)

XSLT Processors

- Web browsers
  - Most web browsers support XSLT 1.0
- XML editors
  - https://en.wikipedia.org/wiki/Comparison_of_XML_editors
  - **XML Copy Editor** https://sourceforge.net/projects/xml-copy-editor
  - **jEdit** (with XML & XSL plugins) http://www.jedit.org
  - **Oxygen XML** https://www.oxygenxml.com/
  - etc.
- Stand-alone XSLT processors
  - https://en.wikipedia.org/wiki/XSLT#Processor_implementations
  - **xsltproc** http://xmlsoft.org/XSLT/
  - etc.

XSLT Stylesheets

XSLT stylesheet is a collection of **template rules**

```
<xsl:stylesheet version="1.0"
  xmlns="http://www.w3.org/1999/XSL/Transform" >
    <xsl:template match="pattern1" >
          ....
    </xsl:template>
                    ⋮
    <xsl:template match="patternn" >
          ....
    </xsl:template>
</xsl:stylesheet>
```

Example 1: students.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <students>
4    <student id="100026">
5      <name>Joe Average</name>
6      <results>
7        <result course="Math 101" grade="C-"/>
8        <result course="Biology 101" grade="C+"/>
9        <result course="Statistics 101" grade="D"/>
10     </results>
11   </student>
12
13   <student id="100078">
14     <name>Jack Doe</name>
15     <results>
16       <result course="Math 101" grade="A"/>
17       <result course="XML 101" grade="A-"/>
18       <result course="Physics 101" grade="B+"/>
19       <result course="XML 102" grade="A"/>
20     </results>
21   </student>
22 </students>
```

Example 1: students.xsl

```
1  <xsl:stylesheet version="1.0"
2                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3
4    <xsl:template match="students">
5      <summary>
6        <xsl:apply-templates select="student"/>
7      </summary>
8    </xsl:template>
9
10   <xsl:template match="student">
11     <grades>
12       <xsl:attribute name="id" select="@id"/>
13       <xsl:apply-templates select=".//@grade"/>
14     </grades>
15   </xsl:template>
16
17   <xsl:template match="@grade">
18     <grade>
19       <xsl:value-of select="."/>
20     </grade>
21   </xsl:template>
22
23 </xsl:stylesheet>
```

Example 1: Output

```
<summary>
        <grades id="100026">
                <grade>C-</grade>
                <grade>C+</grade>
                <grade>D</grade>
        </grades>
        <grades id="100078">
                <grade>A</grade>
                <grade>A-</grade>
                <grade>B+</grade>
                <grade>A</grade>
        </grades>
</summary>
```

Example 2: students.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2
3  <students>
4    <student id="100026">
5      <name>Joe Average</name>
6      <results>
7        <result course="Math 101" grade="C-"/>
8        <result course="Biology 101" grade="C+"/>
9        <result course="Statistics 101" grade="D"/>
10     </results>
11   </student>
12
13   <student id="100078">
14     <name>Jack Doe</name>
15     <results>
16       <result course="Math 101" grade="A"/>
17       <result course="XML 101" grade="A-"/>
18       <result course="Physics 101" grade="B+"/>
19       <result course="XML 102" grade="A"/>
20     </results>
21   </student>
22 </students>
```

Example 2: students2.xsl

```
1  <xsl:stylesheet version="1.0"
2                  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3
4    <xsl:template match="students">
5      <summary>
6        <xsl:apply-templates select="student"/>
7      </summary>
8    </xsl:template>
9
10   <xsl:template match="student">
11     <grades>
12       <xsl:attribute name="id" select="@id"/>
13       <xsl:for-each select=".//@grade">
14             <grade>
15               <xsl:value-of select="."/>
16             </grade>
17       </xsl:for-each>
18     </grades>
19   </xsl:template>
20
21 </xsl:stylesheet>
```

Example 2: Output

```
<summary>
        <grades id="100026">
                <grade>C-</grade>
                <grade>C+</grade>
                <grade>D</grade>
        </grades>
        <grades id="100078">
                <grade>A</grade>
                <grade>A-</grade>
                <grade>B+</grade>
                <grade>A</grade>
        </grades>
</summary>
```

References

- Mozilla's XSLT Basic Example

  https://developer.mozilla.org/en-US/docs/Web/API/XSLTProcessor/Basic_Example
- A. Møller & M. Schwartzbach, An Introduction to XML and Web Technologies, Addison-Wesley, 2006

  http://www.brics.dk/ixwt
- W3C XML Technology https://www.w3.org/standards/xml/
- XML Data Repository at U. Washington

  http://aiweb.cs.washington.edu/research/projects/xmltk/xmldata/