Part VII

Querying XML—The XQuery Data Model

Outline of this part

Querying XML Documents

- "**Querying** XML data" essentially means to
  - **identify (or address) nodes**,
  - to **test** certain further **properties** of these nodes,
  - then to **operate on** the matches,
  - and finally, to **construct result** XML documents as answers.
- In the XML context, the language **XQuery** plays the role that SQL has in relational databases.
- XQuery can express all of the above constituents of XML querying:
  - XPath, as an **embedded sublanguage**, expresses the **locate** and **test** parts;
  - XQuery can then iterate over selected parts, operate on and construct answers from these.
  - There are more XML languages that make use of XPath as embedded sublanguages.
- We will first look into the (XML-based) **data model** used by XQuery and XPath . . .

Motivating example

Recall DilbertML and the comic strip finder:

1. "*Find all bubbles with Wally being angry with Dilbert.*"

   Query: Starting from the root, locate all `bubble` elements somewhere below the `panel` element. Select those `bubble` elements with attributes `@tone` = `"angry"`, `@speaker` = `"Wally"`, and `@to` = `"Dilbert"`.

2. "*Find all strips featuring Dogbert.*"

   Query: Starting from the root, step down to the element `prolog`, then take a step down to element `characters`. Inside the latter, step down to all `character` elements and check for contents being equal to `Dogbert`.

Note the **locate, then test** pattern in both queries.

- An XML parser (with DOM/SAX backend) is all we need to implement such queries.

$\Rightarrow$ **Tedious!** $\Leftarrow$

XPath as an embedded sublanguage

- **XPath**[17] is a declarative, expression-based language to **locate and test** doc nodes (with lots of syntactic sugar to make querying sufficiently sweet).
- Addressing document nodes is a core task in the XML world. XPath occurs as an **embedded sub-language** in
  - **XSLT**[18] (extract and transform XML document [fragments] into XML, XHTML, PDF, ...)
  - **XQuery**[19] (compute with XML nodes of all kinds and their contents, compute new docs, ...)
  - **XPointer**[20] (representation of the address of one or more nodes in a given XML document)

[17] `http://www.w3.org/TR/xpath20/`
[18] `http://www.w3.org/TR/xslt/`
[19] `http://www.w3.org/TR/xquery/`
[20] `http://www.w3.org/TR/xptr/`

The XQuery Data Model

Like for any other database query language, before we talk about the **operators** of the language, we have to specify exactly **what** it is that **these operate on** ...

- XQuery (and the other languages) use an abstract view of the XML data, the so-called **XQuery data model**.

Data Model (DM)

The XQuery DM determines which aspects of an XML document may be inspected and manipulated by an XQuery query.

- What exactly should the XQuery DM look like...?
  
  A simple sequence of characters or other lexical tokens certainly seems inappropriate (too fine-grained)!

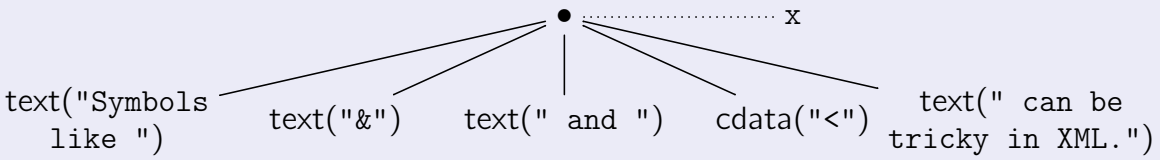
XQuery data model (1)

✎ Which aspects of XML data are relevant to queries?

```
<x>Symbols like &amp; and <![CDATA[<]]> can be
tricky in XML.</x>
```

- What is an adequate representation of XML element `x`?

DOM style. . . ?



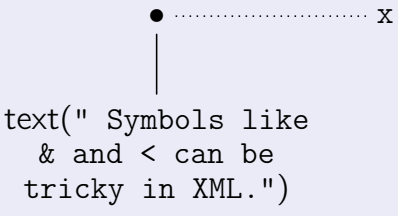
- Faithfully preserves entities and CDATA sections, paying the price of creating more DM nodes during parsing.

XQuery data model (2)

✎ Which aspects of XML data are relevant to queries?

```
<x>Symbols like &amp; and <![CDATA[<]]> can be
tricky in XML.</x>
```

XQuery style. . .



- Do not distinguish between ordinary text, entities, and CDATA sections (the latter two are merely requirements of XML syntax).

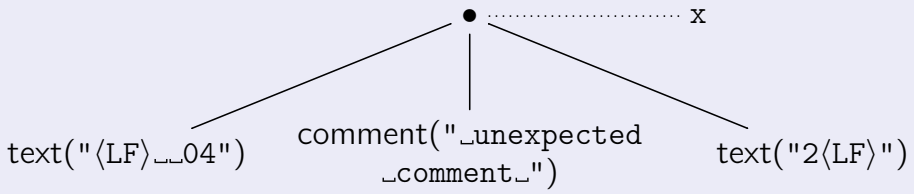
XQuery data model (3): untyped vs. typed

An XML element containing an integer

```
<x>
  04<!-- unexpected comment -->2
</x>
```

⇓

Untyped view . . .



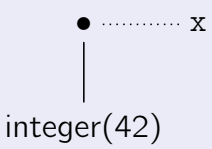
XQuery data model (3): untyped vs. typed

An XML element containing an integer

```
<x>
  04<!-- unexpected comment -->2
</x>
```

⇓

Typed view . . .



- XQuery can work with the typed view, if the input XML document has been **validated** against an XML Schema description.

XQuery DM: Node properties (1)

- A separate W3C document[21] describes the XQuery DM in detail.

In the XQuery DM, an XML element node exhibits a number of properties, including:

node-name	tag name of this element
parent	parent element, may be empty
children	children lists, may be empty
attributes	set of attributes of this element, may be empty
string-value	concatenation of all string values in content
typed-value	element value (after validation only)
type-name	type name assigned by validation

[21] `http://www.w3.org/TR/xpath-datamodel/`

XQuery DM: Node properties (2)

An XML element containing an integer

```
<x>
  04<!-- unexpected comment -->2
</x>
```

⇓

Node properties of unvalidated element `x`

node-name	`x`
parent	$()$
children	(t_1, c, t_2)
attributes	$\emptyset$
string-value	"⟨LF⟩␣␣042⟨LF⟩"
typed-value	"⟨LF⟩␣␣042⟨LF⟩"
type-name	`untypedAtomic`

XQuery DM: Node properties (3)

An XML element containing an integer

```
<x>
  04<!-- unexpected comment -->2
</x>
```

⇓

Node properties of **validated** element `x`

node-name	`x`
parent	$()$
children	(t_1, c, t_2)
attributes	$\emptyset$
string-value	"⟨LF⟩␣␣042⟨LF⟩"
typed-value	42
type-name	`integer`

XQuery: Access to the DM in a query

XQuery provides various ways to access properties of nodes in a query. For example:

access node-name

```
name(<x>content here</x>) ⇒ "x"
```

access parent element (this is actually XPath functionality)

```
<x>content here</x>/parent::* ⇒ ()
```

access string value:

```
string(<x>content here</x>) ⇒ "content here"
```

Items and sequences (1)

Two data structures are pervasive in the XQuery DM:

1. **Ordered, unranked trees of nodes** (XML elements, attributes, text nodes, comments, processing instructions) and
2. **ordered sequences of** zero or more **items**.

✎ Item

An XQuery **item** either is

- a **node** (of one of the kinds listed above), or
- an **atomic value** of one of the 50+ atomic types built into the XQuery DM.

Items and sequences (2)

- A sequence of n items x_1 is written in parentheses, comma-separated

✎ Sequence of length n and empty sequence

$$(x_1, x_2, \ldots, x_n) \qquad ()$$

- A single item x and the singleton sequence (x) are equivalent!
- Sequences *cannot* contain other sequences (*i.e.*, *nested sequences* are implicitly **flattened**):

✎ Flattening, order

```
(0, (), (1, 2), (3))  ≡  (0, 1, 2, 3)
          (0, 1)      ≢  (1, 0)
```

Sequence types (1)

XQuery uses **sequence types** to describe the type of sequences:

Sequence types t (simplified)

$$
\begin{array}{rcl}
t & ::= & \texttt{empty-sequence()} \\
 & | & \textit{item}\ \textit{occ} \\
\textit{occ} & ::= & \texttt{+} \mid \texttt{*} \mid \texttt{?} \mid \varepsilon \\
\textit{item} & ::= & \textit{atomic} \mid \textit{node} \mid \texttt{item()} \\
\textit{node} & ::= & \texttt{element(}\textit{name}\texttt{)} \mid \texttt{text()} \mid \texttt{node()} \mid \cdots \\
\textit{name} & ::= & \texttt{*} \mid \textit{QName} \\
\textit{atomic} & ::= & \texttt{integer} \mid \texttt{string} \mid \texttt{double} \mid \cdots
\end{array}
$$

- A *QName* (qualified name) denotes an element or attribute name, possibly with a name space prefix (*e.g.*, `ns:x`).

Sequence types (2)

Sequence type examples

Value	Sequence type
`42`	`integer, item()`
`<x>foo</x>`	`element(x), item()`
`()`	`empty-sequence(),integer*`
`("foo", "bar")`	`string+, item()*`
`(<x/>, <y/>)`	`element(*)+, node()*`

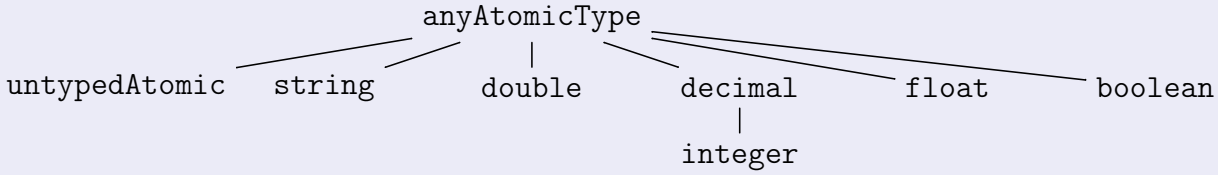
- In the table above, the most specific type is listed first.

Items: atomic values

- XQuery, other than XPath 1.0 or XSLT which exclusively manipulate nodes, can also compute with **atomic values** (numbers, Boolean values, strings of characters, . . .).
  - XQuery knows a rich collection of atomic types (*i.e.*, a versatile hierarchy of number types like fixed and arbitrary precision decimals, integers of different bit-widths, *etc.*).
  - In this course, we will only cover a subset of this rich type hierarchy.
- The hierarchy of atomic types is rooted in the special type `anyAtomicType`.

Hierarchy of atomic types

Atomic Type Hierachy (excerpt)



Numeric literals

```
12345      (: integer :)
12.345     (: decimal :)
12.345E0   (: double  :)
```

Boolean literals

```
true()
false()
```

Computing with untyped values

Atomic values of type `untypedAtomic`, which appear whenever text content is extracted from non-validated XML documents, are **implicitly converted** if they occur in expressions.

Implicit extraction[22] of element content and conversion of values of type `untypedAtomic`

```
                "42" + 1   ⇒  ↯   type error (compile time)
          <x>42</x> + 1    ⇒  43.0E0  (:  double :)
<x>fortytwo</x> + 1        ⇒  ↯   conversion error (runtime)
```

- This behavior saves a lot of explicit casting in queries over non-validated XML documents.

[22] Known as *atomization*, discussed later.

Items: nodes

Just like XML, XQuery differentiates between several **kinds** of nodes:

Six XML node kinds

```
<element attribute="foo">
  text <!--comment-->
  <?processing instruction?>
</element>
```

+ The ("invisible") root node of any complete XML document is the so-called **document node**.

- In XQuery, a query may **extract** and **construct** nodes of all these kinds.

Nodes: observable properties

Each node kind has specific properties but a few important properties are **shared by all kinds:**

✎ Node identity and document order

Each node has a **unique node identity** which is *never* modified. XQuery allows for node identity comparison using the operator `is` .

All nodes are ordered relative to each other, determined by the so-called **document order** (XQuery operator <<). This orders nodes of the same tree according to a *pre-order traversal*.

Nodes in different trees are ordered consistently.

Node identity

Node identity

```
<x>foo</x> is <x>foo</x>    ⇒    false()
```

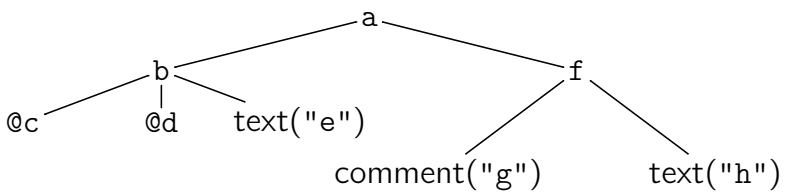
- Note: To compare items based on their **value**, XQuery offers the operators = and `eq`.

Value comparison

```
<x>foo</x> = <x>foo</x>    ⇒    true()
```

Document order

```
<a>
  <b c="..." d="...">e</b>
  <f><!--g-->h</f>
</a>
```



- Parent nodes precede their children and attributes (*e.g.*, `a << b` and `b << @d`). `<<` is transitive.
- Siblings are ordered with attributes coming first (*e.g.*, `b << f`, `@d << text("e")`), but the relative order of attributes (`@c`, `@d`) is implementation-dependent.

Notes on document order

- XML documents always carry this implicit order of their contents.
- Typical XML processing follows this order when accessing components of an XML document (see, *e.g.*, SAX parsing).
- Often, operations on XML documents are supposed to deliver their results also in this particular order. Document order is part of the (formal) semantics of many XML related languages.
- Contrast this with relational database query languages, where **set-orientation** always gives the freedom to the query processor to access and deliver tuples in arbitrary order!
- We will (later) see that document order has far-reaching consequences XML query processing.