XML query languages

XPath
XQuery

XPath

- XPath is a language for describing paths in XML documents.
  - Think of an SSD graph and *its* paths.
- Path descriptors are similar to path descriptors in a (UNIX) file system.
  - A simple path descriptor is a sequence of element names separated by slashes (/).
  - / denotes the root of a document.
  - // means the path can start anywhere in the tree from the current node.

Examples:

```
<Courses>
 <Course name="Databases" code="TDA357">
  <GivenIn period="2" teacher="Niklas Broberg" />
  <GivenIn period="4" teacher="Rogardt Heldal" />
 </Course>
 <Course name="Algorithms" code="TIN090">
  <GivenIn period="1" teacher="Devdatt Dubhashi" />
 </Course>
</Courses>
```

`/Courses/Course/GivenIn` will return the set of all **`GivenIn`** elements in the document.

`//GivenIn` will return the same set, but only since we know by our schema that **`GivenIn`** elements can only appear in that position.

`/Courses` will return the document as it is.

More path descriptors

- There are other path descriptors than / and //:
  - **`*`** denotes any one element:
    - **`/Courses/*/*`** will give all children of all children of a **`Courses`** element, i.e. all **`GivenIn`** elements.
    - **`//*`** will give all elements anywhere.
  - **`.`** denotes the current element:
    - **`/Courses/Course/.`** will return the same elements as **`/Courses/Course`**
  - **`..`** denotes the parent element:
    - **`//GivenIn/..`** will return all elements that have a **`GivenIn`** element as a child.
- Think about how we can traverse the graph – upwards, downwards, along labelled edges etc.

Attributes

- Attributes are denoted in XPath with a `@` symbol:
  - **`/Courses/Course/@name`** will give the names of all courses.

Quiz: For the Scheduler example, what will the path expression **`//@name`** result in?

The names of all courses, and the names of all rooms.

Axes

- The various directions we can follow in a graph are called *axes* (sing. axis).
- General syntax for following an axis is

  ***`axis`*`::`**

  – Example: **`/Courses/child::Course`**
- Only giving a label is shorthand for **`child::label`**, while `@` is short for **`attribute::`**

More axes

- Some other useful axes are:
  - parent:: = parent of the current node.
    - Shorthand is ..
  - descendant-or-self:: = the current node(s) and all descendants (i.e. children, their children, …) down through the tree.
    - Shorthand is //
  - ancestor::, ancestor-or-self = up through the tree
  - following-sibling:: = any elements on the same level that come *after* this one.
  - …

Selection

- We can perform tests in XPath expressions by placing them in square brackets:
  - **`/Courses/Course/GivenIn[@period = 2]`** will give all **`GivenIn`** elements that regard the second period.

Quiz: What will the path expression
`/Courses/Course[GivenIn/@period = 2]`

result in?

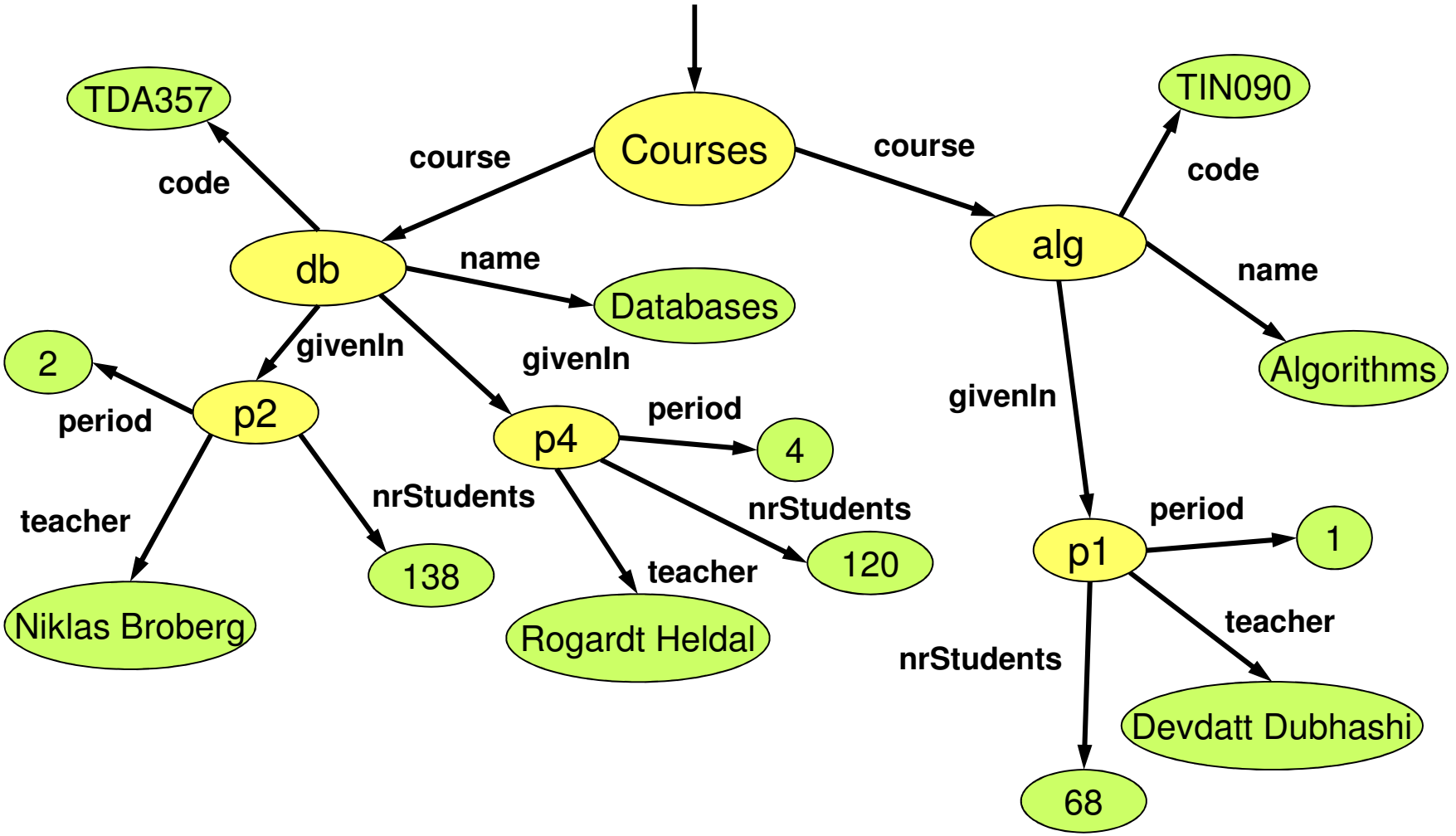
All **`Course`** elements that are given in the second period (but for each of those, all the **`GivenIn`** elements for that course).

Quiz!

Write an XPath expression that gives the courses that are given in period 2, but with only the GivenIn element for period 2 as a child!

It can't be done!
XPath is not a full query language, it only allows us to specify paths to elements or groups of elements. We can restrict in the path using [ ] notation, but we cannot restrict further down in the tree than what the path points to.

Example: /**Courses/Course[GivenIn/@period = 2]**



XQuery

- XQuery is a full-fledged querying language for XML documents.
  - Cf. SQL queries for relational data.
- XQuery is built on top of XPath, and uses XPath to point out element sets.

- XQuery is a W3 recommendation.

XQuery “Hello World”

If our XQuery file contains:

```
<Greeting>Hello World</Greeting>
```

or:

```
let $s := "Hello World"
return <Greeting>{$s}</Greeting>
```

then the XQuery processor will produce the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<Greeting>Hello World</Greeting>
```

Function doc("file.xml")

```
bash$ cat example.xq
doc("courses.xml")
bash$ xquery example.xq
<?xml version="1.0" encoding="UTF-8"?>
<Courses>
  <Course name="Databases" code="TDA357">
    <GivenIn period="2" teacher="Niklas Broberg"/>
    <GivenIn period="4" teacher="Rogardt Heldal"/>
  </Course>
  <Course name="Algorithms" code="TIN090">
    <GivenIn period="1" teacher="Devdatt Dubhashi"/>
  </Course>
</Courses>
```

Quiz!

Write an XQuery expression that puts extra <Result></Result> tags around the result, e.g.

```
<Result>
  <Courses>
    <Course name="Databases" code="TDA357">
      <GivenIn period="2" teacher="Niklas Broberg"/>
      <GivenIn period="4" teacher="Rogardt Heldal"/>
    </Course>
    <Course name="Algorithms" code="TIN090">
      <GivenIn period="1" teacher="Devdatt Dubhashi"/>
    </Course>
  </Courses>
</Result>
```

Putting tags around the result

Curly braces are necessary to evaluate the expression between the tags.

```
<Result>{doc("courses.xml")}</Result>
```

Alternatively, we can use a **let** clause to assign a value to a variable. Again, curly braces are needed to get the value of variable $d.

```
let $d := doc("courses.xml")
return <Result>{$d}</Result>
```

FLWOR

- Basic structure of an XQuery expression is:
  - FOR-LET-WHERE-ORDER BY-RETURN.
  - Called FLWOR expressions (pronounce as *flower*).
- A FLWOR expression can have any number of FOR (iterate) and LET (assign) clauses, possibly mixed, followed by possibly a WHERE clause and possibly an ORDER BY clause.
- Only required part is RETURN.

Quiz!

What does the following XQuery expression compute?

```
let $courses := doc("courses.xml")
for $gc in $courses//GivenIn
where $gc/@period = 2
return <Result>{$gc}</Result>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Result>
  <GivenIn period="2" teacher="Niklas Broberg"/>
</Result>
```

Quiz!

What does the following XQuery expression compute?

```
let $courses := doc("courses.xml")
let $gc := $courses//GivenIn[@period = 2]
return <Result>{$gc}</Result>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Result>
  <GivenIn period="2" teacher="Niklas Broberg"/>
</Result>
```

Quiz!

What does the following XQuery expression compute?

```
let $courses := doc("courses.xml")
for $c in $courses/Courses/Course
let $code := $c/@code
let $given := $c/GivenIn
where $c/GivenIn/@period = 2
return <Result code="{$code}">{$given}</Result>
```

```
<? xml version="1.0" encoding="UTF-8"?>
<Result code="TDA357">
  <GivenIn period="2" teacher="Niklas Broberg"/>
  <GivenIn period="4" teacher="Rogardt Heldal"/>
</Result>
```

Quiz!

Write an XQuery expression that gives the courses that are given in period 2, but with only the **`GivenIn`** element for period 2 as a child!

```
let $courses := doc("courses.xml")
for $c in $courses/Courses/Course
let $code := $c/@code, $name := $c/@name
let $gc := $c/GivenIn[@period = 2]
where not(empty($gc))
return <Course code="{$code}"
              name="{$name}">{$gc}</Course>
```

A sequence of elements

The previous examples have all returned a single element. But an XQuery expression can also evaluate to a sequence of elements, e.g.

```
let $courses := doc("courses.xml")
for $gc in $courses/Courses/Course/GivenIn
return $gc
```

```
<GivenIn period="2" teacher="Niklas Broberg"/>
<GivenIn period="4" teacher="Rogardt Heldal"/>
<GivenIn period="1" teacher="Devdatt Dubhashi"/>
```

Putting tags around a sequence

```
let $courses := doc("courses.xml")
let $seq := (
    for $gc in $courses/Courses/Course/GivenIn
    return $gc )
return <Result>{$seq}</Result>
```

```
<Result>
  {
    let $courses := doc("courses.xml")
    for $gc in $courses/Courses/Course/GivenIn
    return $gc
  }
</Result>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Result>
  <GivenIn period="2" teacher="Niklas Broberg"/>
  <GivenIn period="4" teacher="Rogardt Heldal"/>
  <GivenIn period="1" teacher="Devdatt Dubhashi"/>
</Result>
```

Cartesian product

Two **for** clauses will iterate over all combinations of values for the loop variables, e.g.

```
let $courses := doc("courses.xml")
for $c in $courses/Courses/Course
for $gc in $courses/Courses/Course/GivenIn
return <Info name="{$c/@name}" teacher="{$gc/@teacher}" />
```

```
<Info name="Databases" teacher="Niklas Broberg"/>
<Info name="Databases" teacher="Rogardt Heldal"/>
<Info name="Databases" teacher="Devdatt Dubhashi"/>
<Info name="Algorithms" teacher="Niklas Broberg"/>
<Info name="Algorithms" teacher="Rogardt Heldal"/>
<Info name="Algorithms" teacher="Devdatt Dubhashi"/>
```

Aggregations

XQuery provides the usual aggregation functions, count, sum, avg, min, max.

```
<Result>
  {
    count(doc("scheduler.xml")//Room)
  }
</Result>
```

```
<Result>
  {
    sum(doc("scheduler.xml")//Room/@nrSeats)
  }
</Result>
```

Joins in XQuery

We can join two or more documents in XQuery by calling the function doc() two or more times.

```
let $a = doc("a.xml")
let $b = doc("b.xml")
...
   (... compare values in $a with values in $b ...)
```

Quiz: what does this XQuery expression compute?

```
<Result>
  {
    for $d in ( doc("scheduler.xml"), doc("courses.xml") )
    return $d
  }
</Result>
```

Sorting in XQuery

```
<Result>
  {
    let $courses := doc("courses.xml")
    for $gc in $courses/Courses/Course/GivenIn
    order by $gc/@period
    return $gc
  }
</Result>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Result>
  <GivenIn period="1" teacher="Devdatt Dubhashi"/>
  <GivenIn period="2" teacher="Niklas Broberg"/>
  <GivenIn period="4" teacher="Rogardt Heldal"/>
</Result>
```

Eliminating duplicates

```
<Scheduler>
 <Courses>
  <Course code="TDA357" name="Databases">
   <GivenIn period="2" teacher="Graham Kemp">
    <Lecture weekday="Tuesday"  hour="10:00" room="HB2" />
    <Lecture weekday="Friday"   hour="10:00" room="HB2" />
   </GivenIn>
   <GivenIn period="3" teacher="Niklas Broberg">
    <Lecture weekday="Monday"   hour="15:15" room="VR" />
    <Lecture weekday="Thursday" hour="10:00" room="HB1" />
   </GivenIn>
  </Course>
 </Courses>
</Scheduler>
```

Find rooms where lectures have been scheduled (sorted by room name, and without duplicates).

Eliminating duplicates

```
<Result>
  {
    let $s := doc("scheduler.xml")
    for $r in distinct-values($s//Lecture/@room)
    order by $r
    return <Room name="{$r}" />
  }
</Result>
```

```
<Result>
  <Room name="HB1"/>
  <Room name="HB2"/>
  <Room name="VR"/>
</Result>
```

if-then-else expression

```
<Result>
  {
    for $r in doc("scheduler.xml")//Room
    return
        if ( $r/@nrSeats > 200 )
        then <BigRoom name="{$r/@name}" />
        else <SmallRoom name="{$r/@name}" />
  }
</Result>
```

```
<Result>
  <BigRoom name="VR"/>
  <SmallRoom name="HB1"/>
</Result>
```

Quantification in XQuery

An XQuery expression might evaluate to a single item or a sequence of items.

```
every variable in expression satisfies condition

 some variable in expression satisfies condition
```

Most tests in XQuery, such as the "=" comparison operator, are existentially quantified anyway, so "some" is rarely needed.

Comparing items in XQuery

- The comparison operators eq, ne, lt, gt, le and ge can be used to compare single items.
- If either operand is a sequence of items, the comparison will fail.

Updating XML

- We have corresponding languages for XML and relational databases:
  - SQL DDL ⇔ DTDs or XML Schema.
  - SQL queries ⇔ XQuery
  - SQL modifications ⇔ ??
- XQuery Update Facility 1.0 is a W3C recommendation (March 2011)
  - insert, delete, replace, rename, transform expressions

Warning …

- “Many companies report a strong interest in XML. XML however, is so flexible that this is similar to expressing a strong interest in ASCII characters.”

  http://xml.coverpages.org/BiztalkFrameworkOverviewFinal.html

Looking to the future

  – RDF, RDF Schema, OWL, …

Summary XML

- XML is used to describe data organized as *documents*.
  - Semi-structured data model.
  - Elements, tags, attributes, children.
  - Namespaces.
- XML can be valid with respect to a schema.
  - DTD: ELEMENT, ATTLIST, CDATA, ID, IDREF
  - XML Schema: Use XML for the schema domain to describe your schema.
- XML can be queried for information:
  - XPath: Paths, axes, selection
  - XQuery: FLWOR.

Exam –XML

”A medical research facility wants a database that uses a semi-structured model to represent different degrees of knowledge regarding the outbreak of epidemic diseases. …”

- Suggest how to model this domain as a DTD (or XML Schema).
- Discuss the benefits of the semi-structured data model for this particular domain.
- Given this DTD, what does this XPath/XQuery expression compute?
- Write an XQuery expression that computes…