XML and Xpath

Literature: KBL 17.1-17.2.3, 17.4

Rasmus Pagh

Serialization

- How to store information in **serial form** (e.g. in a file, or for transmission)?
  - Relations can be stored using comma-separated values (CSV), or similar.
  - Ad-hoc formats (e.g. most older text processing and spreadsheet formats)
  - Grammar-based formats: E.g. programs, human-edited data files (IMDB), …
    - BSWU: More in ”Programmer som data”.
  - …

Formats are controversial

"You sent the attachment in Microsoft Word format, a secret proprietary format, so I cannot read it. If you send me the plain text, HTML, or PDF, then I could read it."

Richard L. Stallman

Formats are controversial

From Wikipedia:

The ISO standardization of Office Open XML was controversial and embittered,[19] with much discussion both about the specification and about the standardization process.[20] According to InfoWorld:

> OOXML was opposed by many on grounds it was unneeded, as software makers could use OpenDocument Format (ODF), a less complicated office software format that was already an international standard.[19]

...

IBM (which supports the ODF format) threatened to leave standards bodies ...

The dream…

One universal standard for communication

- http://www.youtube.com/watch?v=6ptQGX__rd0

The format zoo

- Not all data is in relational databases!
  - Files (.txt, .html, .doc, .xls, .cs, .jar, .ser,…)
  - Services (e.g. web servers, file transmission)
  - Software-to-software communication
  - Data streams (e.g. audio/video streams)

- Useful with a **framework**:
  - Gives common language for describing data.
  - Allows common **tools** (akin to an RDBMS)
- Widespread framework: **XML**.

Today’s lecture

- What is an XML file?
- What are namespaces?
- XML tools, part 1:
  - Parsers (SAX, DOM)
  - Xpath query language

Compression

- An important aspect of serialization is *compression* (want small files, fast transmission,…)
- Traditionally, compression was often done in ad-hoc ways (e.g. encode 8 yes/no values as 1 byte).
- Modern approaches view compression as **orthogonal** to logical encoding
  - Can be applied as post-processing
  - Examples: jar files, docx format, …

An Introduction to XML and Web Technologies

XML Documents

the following slides are based on slides by
Anders Møller & Michael I. Schwartzbach
© 2006 Addison-Wesley

What is XML?

- XML: *E**x**tensible **M**arkup **L**anguage*
- A **framework** for defining "markup languages" (e.g. (X)HTML)
- Each language is targeted at its own **application domain** with its own markup tags
- There is a common set of **generic tools** for processing XML documents
- Inherently **internationalized** and **platform independent** (Unicode)
- Developed by W3C, standardized in 1998

xkcd take on standards

Case study: Recipes in XML

- Define our own "***Recipe Markup Language***"
- Choose markup tags that correspond to concepts in this application domain
  - *recipe*, *ingredient*, *amount*, …

Example (1/2)

```
<collection>
  <description>Recipes suggested by Jane Dow</description>

  <recipe id="r117">
    <title>Rhubarb Cobbler</title>
    <date>Wed, 14 Jun 95</date>

    <ingredient name="diced rhubarb" amount="2.5" unit="cup"/>
    <ingredient name="sugar" amount="2" unit="tablespoon"/>
    <ingredient name="fairly ripe banana" amount="2"/>
    <ingredient name="cinnamon" amount="0.25" unit="teaspoon"/>
    <ingredient name="nutmeg" amount="1" unit="dash"/>

    <preparation>
      <step>
        Combine all and use as cobbler, pie, or crisp.
      </step>
    </preparation>
```

Example (2/2)

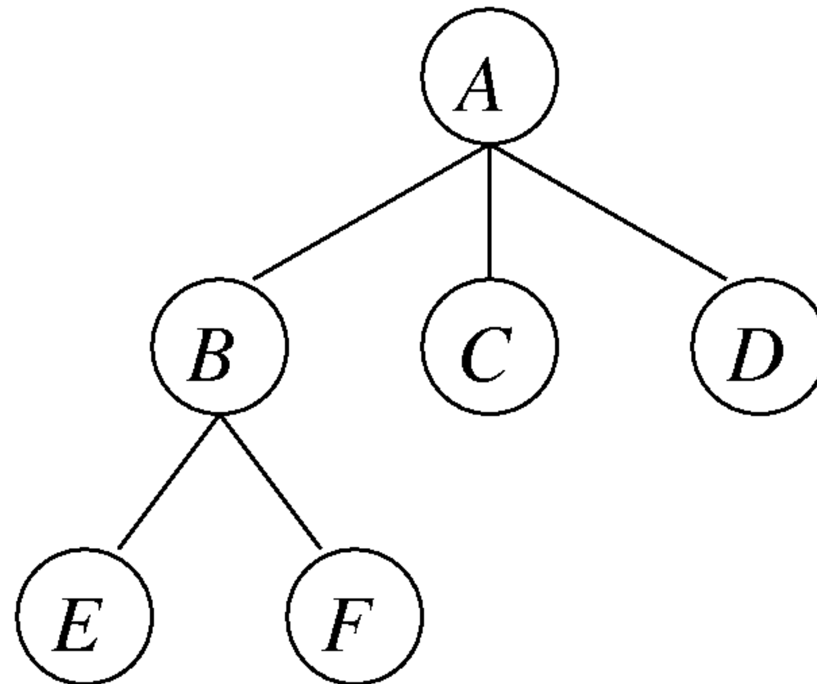
```
    <comment>
      Rhubarb Cobbler made with bananas as the main sweetener.
      It was delicious.
    </comment>

    <nutrition calories="170" fat="28%"
               carbohydrates="58%" protein="14%"/>
    <related ref="42">Garden Quiche is also yummy</related>
  </recipe>
</collection>
```

Many web browsers are good XML viewers

XML Trees

- Conceptually, an XML document is a **tree structure**
  - node, edge
  - root, leaf
  - child, parent
  - sibling (ordered), ancestor, descendant



- **Terminology**: element = node

An Analogy: File Systems

```
                /
        bin/          home/
emacs   grep   bash   jdoe/                 jsmith/
               work/       fun/        letters/    paper.pdf
Hello.java  Hello.class  holiday.jpg   Hello.doc
```

Tree View of the XML Recipes

collection

description

Recipes suggested by Jane Dow

recipe

id: 117

title

date

ingredient

...

ingredient

preparation

comment

nutrition

related

Rhubarb Cobbler

Wed, 14 Jun 95

name: diced rhubarb
amount: 2.5
unit: cup

name: nutmeg
amount: 1
unit: dash

step

Combine all and use as cobbler, pie, or crisp.

Rhubarb Cobbler made with bananas as the main sweetener. It was delicious.

calories: 170
fat: 28%
carbohydrates: 58%
protein: 14%

ref: 42

Garden Quiche is also yummy

XML parsers

- A basic tool for processing XML is a *parser* that reads well-formed XML and presents it in a way that makes it easy to work with.
- Two main types:
  - Event-driven (SAX API): simply reports the tags it sees (may call user-defined methods).
  - Parse tree (DOM API): construct a ”parse tree” of the XML document.

Node types in XML trees

- **Text nodes**: written as the text they carry
- **Element nodes**: start-end tags
  - <*bla* ...> ... </*bla*>
  - short-hand notation for empty elements: <*bla*/>
- **Attribute nodes**: *name*="*value*" in start tags
- **Comment nodes**: <!-- *bla* -->
- **Processing instructions**: <?*target value*?>

Well-formedness

- Every XML document must be ***well-formed***
  - start and end tags must **match** and **nest** properly
    - <x><y></y></x> ✓
    - ~~</z><x><y></x></y>~~
  - exactly one **root element**
  - ...
- in other words, it defines a proper tree structure

Example: XHTML

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head><title>Hello world!</title></head>
  <body>
    <h1>This is a heading</h1>
    This is some text.
  </body>
</html>
```

Problem session

- Think about how one would represent a general XML document in a relational database.
- Come up with at least one natural query on XML data that would be difficult to write using standard SQL.

XML Namespaces

```
<widget type="gadget">
  <head size="medium"/>
  <big><subwidget ref="gizmo"/></big>
  <info>
    <head>
      <title>Description of gadget</title>
    </head>
    <body>
      <h1>Gadget</h1>
      A gadget contains a big gizmo
    </body>
  </info>
</widget>
```

- When combining languages, element names may become **ambiguous**!

The Idea

- Assign a URI to every (sub-)language
  e.g. `http://www.w3.org/1999/xhtml`
  for XHTML 1.0

- Qualify element names with URIs:

  {http://www.w3.org/1999/xhtml}`head`

The Actual Solution

- *Namespace declarations* bind URIs to *prefixes*

```
<... xmlns:foo="http://www.w3.org/TR/xhtml1">
   ...
   <foo:head>...</foo:head>
   ...
</...>
```

- Lexical scope (like java and C#)
- Default namespace (no prefix) declared with `xmlns="..."`
- Attribute names can also be prefixed

Widgets with Namespaces

```
<widget type="gadget" xmlns="http://www.widget.inc">
  <head size="medium"/>
  <big><subwidget ref="gizmo"/></big>
  <info xmlns:xhtml="http://www.w3.org/TR/xhtml1">
    <xhtml:head>
      <xhtml:title>Description of gadget</xhtml:title>
    </xhtml:head>
    <xhtml:body>
    <xhtml:h1>Gadget</xhtml:h1>
     A gadget contains a big gizmo
    </xhtml:body>
  </info>
</widget>
```

An Introduction to XML and Web Technologies

The XPath Language

the following slides are based on slides by
Anders Møller & Michael I. Schwartzbach
© 2006 Addison-Wesley

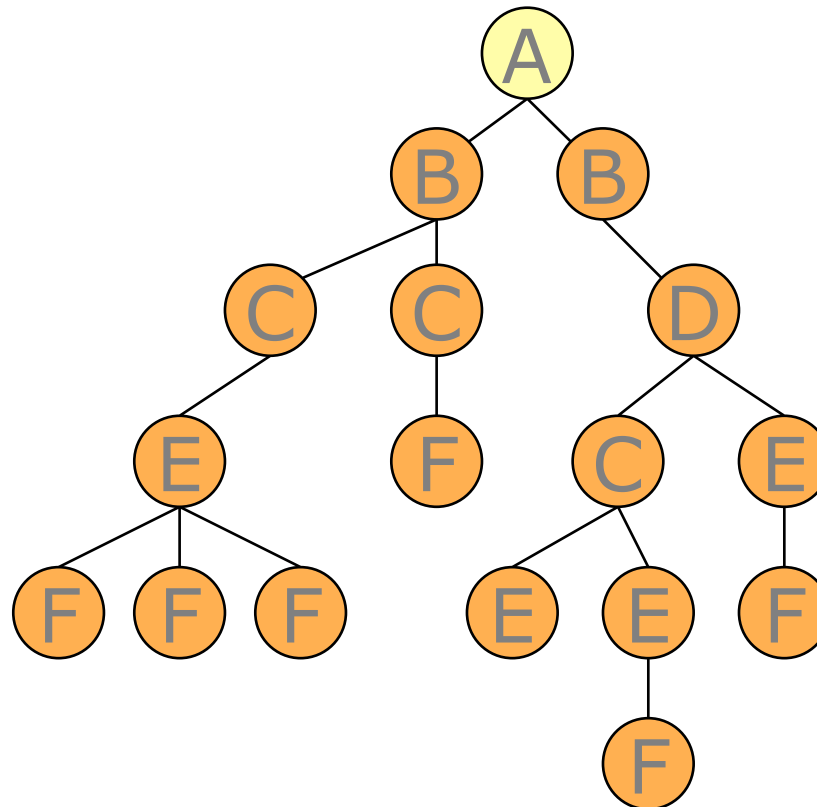
XPath Expressions

- Flexible notation (a simple query language) for **navigating** around trees
- A basic technology that is **widely used** in other XML languages (e.g. XSLT and XQuery).
- Simple Xpath expression similar to ways of listing files:
  - `/teaching/*/*/recipes`
  - `/teaching/*/xml/../pensum`

Xpath Location Paths

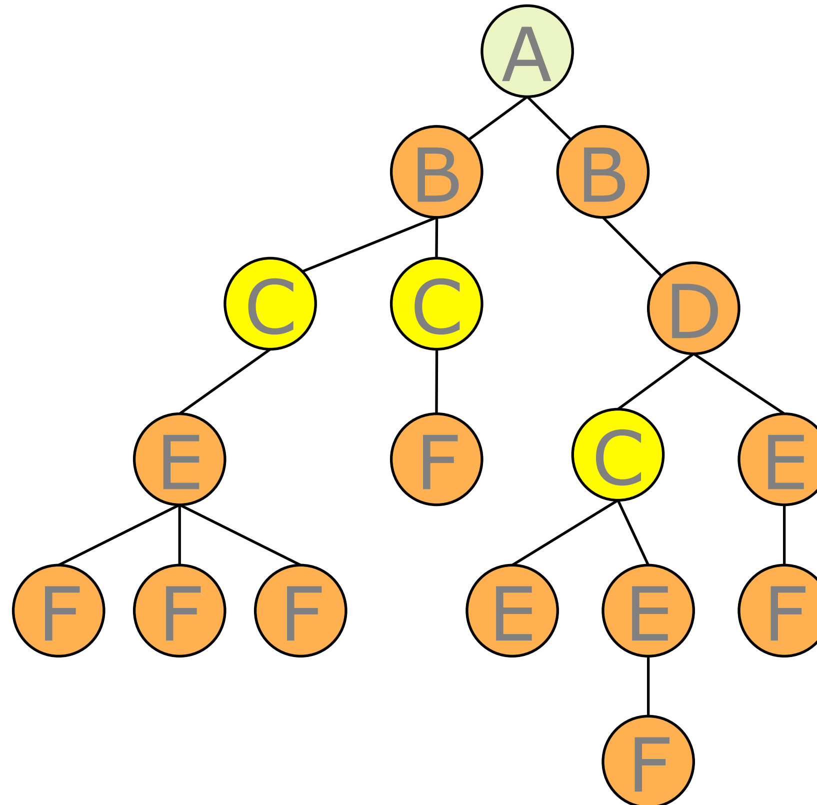
- A *location path* evaluates to a **sequence** of nodes
  - Intuitively, the set of nodes that can be reached by following the path(s) described.
- The sequence is **sorted** in ”document order” (start tag position).
- The sequence will **never** contain **duplicates**

An Example



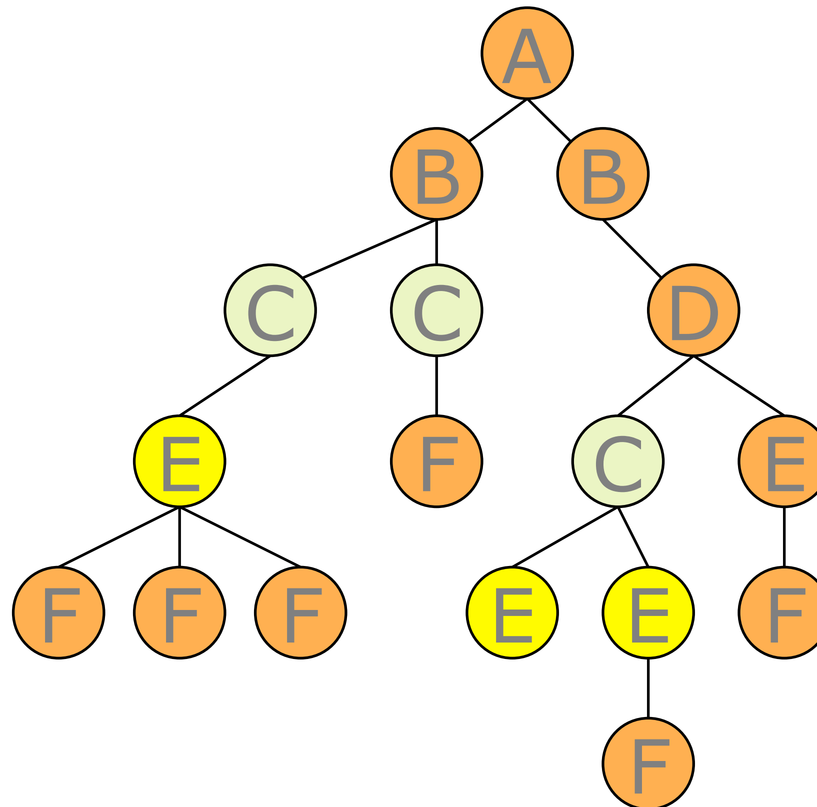
Context node

An Example



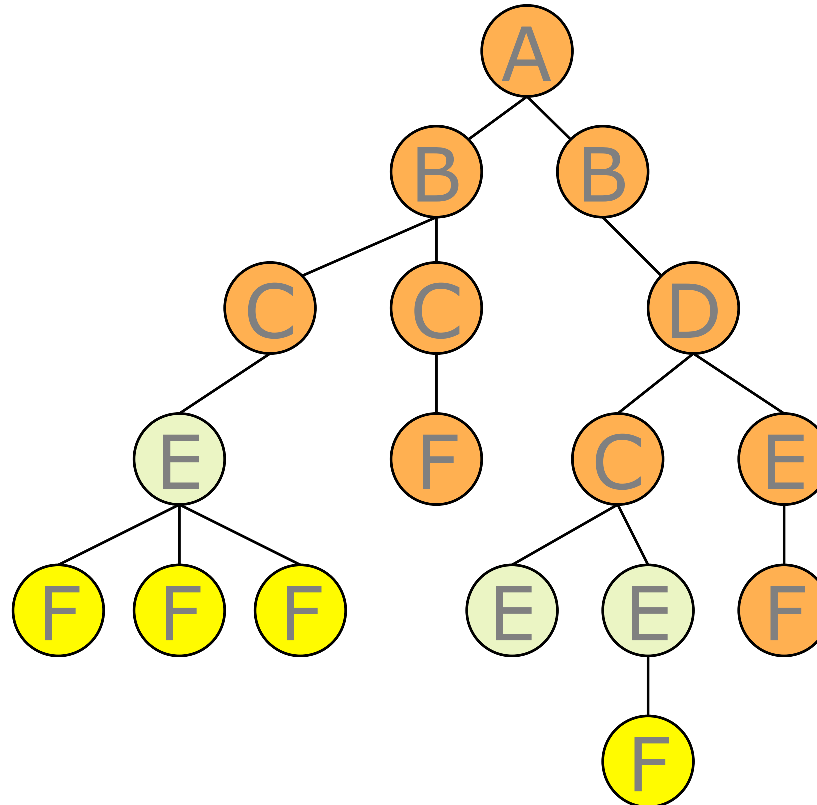
//C

An Example



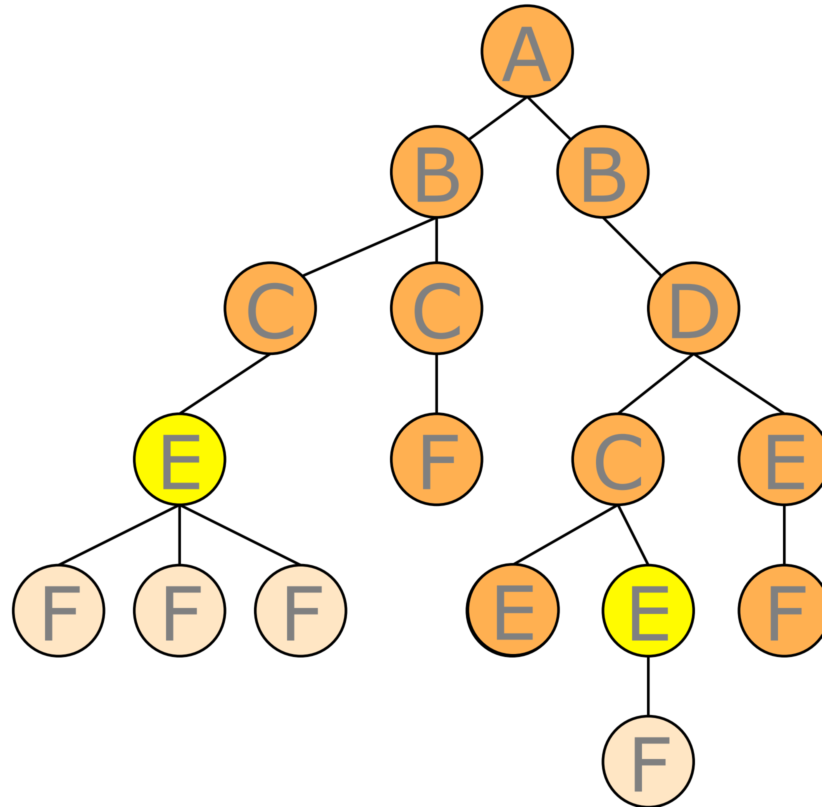
//C/E

An Example



//C/E/*

An Example



//C/E/*/..

An Example

//C/E/*/..[2]

Axes

- XPath expressions work by specifying a sequence of movements along ”axes”.

- XPath supports 12 different axes
  - child: /
  - descendant: //
  - parent: ..
  - ancestor
  - following-sibling
  - preceding-sibling
  - Attribute: /@
  - following
  - preceding
  - self
  - descendant-or-self
  - ancestor-or-self

The parent Axis

The `child` Axis

The descendant Axis

The ancestor Axis

The `following` Axis

The `preceding` Axis

Location Steps

- XPath expressions are made of a sequence of *location steps*
- A *location step* consists of
  - an *axis*
  - a *nodetest* (node name or *)
  - optionally, some *predicates* (in square brackets)

  axis :: *nodetest* [Exp_1] [Exp_2] …
- Semantics:
  - Apply the steps one at a time, starting with the root element.
  - A step produces the **union** of steps applied to results of the previous step.

eXist demo

- Install eXist XML database, or go to
  http://exist.itu.dk:8080/exist/sandbox/sandbox.xql
- Upload an XML document, and start writing Xpath / XQuery!
  - Well, *almost* that simple…
- **Example**: `//preparation/step[1]`
  (matches all first step elements inside preparation elements).
- With namespace wrapper:

```
declare default element namespace
   ”http://www.brics.dk/ixwt/recipes”;
doc("recipes.xml")//preparation/step[1]
```

eXist tips

- For queries that return attributes, must “wrap” the result in an XML element (otherwise there is a silent error).
- **Example**:

```
<a>{
doc("recipes.xml")/(//ingredient)[4]/@name
}</a>
```

- If the *text* of an attribute is desired, use the `string()` function, e.g.:

```
doc("recipes.xml”)//ingredient/@name/string()
```

Predicates, examples

- Name of ingredients measured in cups:

```
//ingredient[@unit='cup']/@name
```

- All first ingredients:

```
//ingredient[1]/@name
```

- Ingredients containing ingredients:

```
//ingredient[//ingredient]/@name
```

Predicates

- Can be general XPath expressions – result converted into a boolean.
- Evaluated with the current node as context
- Result is coerced into a boolean
  - a number yields true if it equals the context position
  - a string yields true if it is not empty
  - a sequence yields true if it is not empty

Problem session

- Consider the recipe collection. Write Xpath for:
  - Finding the name of all ingredients.
  - Finding the ingred. names for Rhubarb Cobbler.
  - Finding the titles of recipes that contain sugar.

```
<collection>
  <description>Recipes suggested by Jane Dow</description>

  <recipe id="r117">
    <title>Rhubarb Cobbler</title>
    <date>Wed, 14 Jun 95</date>

    <ingredient name="diced rhubarb" amount="2.5" unit="cup"/>
    <ingredient name="sugar" amount="2" unit="tablespoon"/>
    <ingredient name="fairly ripe banana" amount="2"/>
    <ingredient name="cinnamon" amount="0.25" unit="teaspoon"/>
    <ingredient name="nutmeg" amount="1" unit="dash"/>

    <preparation>
      <step>
        Combine all and use as cobbler, pie, or crisp.
      </step>
    </preparation>
```

Value Comparison

- Operators: eq, ne, lt, le, gt, ge
- Natural semantics when used on atomic values

```
8 eq 4+4
(//rcp:ingredient)[1]/@name eq ”beef cube steak”
```

- Two XML elements can be compared for equality using the is operator.

General Comparison

- Operators: =, !=, <, <=, >, >=
- When used on a sequence of atomic values:
  - if there **exists** two values, one from each argument, where the comparison holds, the result is true
  - otherwise, the result is false

```
8 = 4+4
(1,2) = (2,4)
//rcp:ingredient/@name = ”salt”
```

Be Careful About Comparisons

```
((//rcp:ingredient)[4]/@name,(//rcp:ingredient)[4]/@amount) eq
(//rcp:ingredient)[5]/@name, (//rcp:ingredient)[5]/@amount))
```

Yields false, since the arguments are not singletons

```
((//rcp:ingredient)[40]/@name, (//rcp:ingredient)[41]/@amount) =
((//rcp:ingredient)[53]/@name, (//rcp:ingredient)[54]/@amount
```

Yields true, since two names are found to be equal

```
((//rcp:ingredient)[4]/@name, (//rcp:ingredient)[4]/@amount) is
((//rcp:ingredient)[5]/@name, (//rcp:ingredient)[5]/@amount)
```

Yields a runtime error, since the arguments are not singletons

XPath violates usual math rules

- Reflexivity?

  `()=()` yields false

- Transitivity?

  `(1,2)=(2,3)`, `(2,3)=(3,4)`, not `(1,2)=(3,4)`

- Anti-symmetry?

  `(1,4)<=(2,3)`, `(2,3)<=(1,4)`, not `(1,2)=(3,4)`

- Negation?

  `(1)!=()` yields false, `(1)=()` yields false

Functions

- XPath has an extensive *function library*, Examples*:* `fn:count` *and* `fn:not`*.*
- Default *namespace* for functions:
  `http://www.w3.org/2006/xpath-functions`
- 106 functions are required.

- Overview of functions:
  `http://www.w3schools.com/Xpath/xpath_functions.asp`

`for` expressions

- Collects results using iteration. E.g.:

```
for $r in //rcp:recipe
  return fn:count($r//rcp:ingredient[fn:not(rcp:ingredient)])
```

returns the value

```
11, 12, 15, 8, 30
```

- The expression

```
 for $i in (1 to 5)
   for $j in (1 to $i)
     return $j
```

returns the value

```
 1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5
```

XML/relational integration

- The easy part: Import/export relations in XML format.
  - Most DBMSs do this, MySQL from ver. 5.1.
- Harder: Support XML as a data type.
  - Easier way: XML is a string (MySQL)
  - Harder way: ”Native” support
- Hardest: Integrate SQL and (e.g.) Xpath
  - Proposed mechanism: SQL/XML
  - Rival (?): Xquery (SQL-like XML queries, next week)

XML in MySQL

- Support for a limited subset of XPath.
- XML is treated as text values, queried through the function ExtractValue (which returns a string).
- **Example**:

```
select id,ExtractValue(descr,'//rcp:title')
from recipelist;
```

- XML as an *export* format: Will be used in last hand-in (due in 2 weeks).

Summary

- XML is a framework for representing data in a ”markup language”.
- Namespaces is a mechanism for making element names globally unique.
- XML comes with a number of tools:
  - Parsers (SAX, DOM)
  - XPath interpreters (used as sublanguage)
  - More next week…

More XPath

- The following slides give more information and examples on XPath.
- They are part of the course curriculum and can be considered supplements to the literature in XPath.

General Expressions

- Every Xpath expression evaluates to a *sequence* of
  - *atomic values, or*
  - *nodes*
- Atomic values may be
  - *numbers*
  - *booleans*
  - Unicode *strings*
- Nodes have *identity*

Atomization

- A sequence may be *atomized*
- This results in a sequence of *atomic values*
- For element nodes this is the *concatenation* of all descendant *text nodes*
- For other nodes this is the *obvious string*

Sequence Expressions

- The ’,’ operator concatenates sequences
- Integer ranges are constructed with ’`to`’
- Operators: `union`, `intersect`, `except`
- Sequences are always *flattened*
- These expressions give the same result:

```
(1,(2,3,4),((5)),(),(((6,7),8,9)))
1 to 9
1,2,3,4,5,6,7,8,9
```

Filter Expressions

- Predicates generalized to *arbitrary* sequences
- The expression ’.’ is the *context item*
- The expression:

```
(10 to 40)[. mod 5 = 0 and position()>19]
```

has the result:

```
30, 35, 40
```

Value Comparison

- Operators: `eq`, `ne`, `lt`, `le`, `gt`, `ge`
- Used on atomic values
- When applied to arbitrary values:
  - atomize
  - if either argument is empty, the result is empty
  - if either has length >1, the result is false
  - if incomparable, a runtime error
  - otherwise, compare the two atomic values

```
8 eq 4+4
(//rcp:ingredient)[1]/@name eq ”beef cube steak”
```

General Comparison

- Operators: =, !=, <, <=, >, >=
- Used on general values:
  - atomize
  - if there **exists** two values, one from each argument, whose comparison holds, the result is true
  - otherwise, the result is false

```
8 = 4+4
(1,2) = (2,4)
//rcp:ingredient/@name = ”salt”
```

Example Functions

```
fn:abs(-23.4) = 23.4
fn:ceiling(23.4) = 24
fn:floor(23.4) = 23
fn:round(23.4) = 23
fn:round(23.5) = 24
```

```
fn:exists(()) = fn:false()
fn:exists((1,2,3,4)) = fn:true()
fn:empty(()) = fn:true()
fn:empty((1,2,3,4)) = fn:false()
fn:count((1,2,3,4)) = 4
fn:count(//rcp:recipe) = 5
```

```
fn:not(0) = fn:true()
fn:not(fn:true()) = fn:false()
fn:not("") = fn:true()
fn:not((1)) = fn:false()
```

More Example Functions

```
fn:concat("X","ML") = "XML"
fn:concat("X","ML"," ","book") = "XML book"
fn:string-join(("XML","book")," ") = "XML book"
fn:string-join(("1","2","3"),"+") = "1+2+3"
fn:substring("XML book",5) = "book"
fn:substring("XML book",2,4) = "ML b"
fn:string-length("XML book") = 8
fn:upper-case("XML book") = "XML BOOK"
fn:lower-case("XML book") = "xml book"
```

```
fn:avg((2, 3, 4, 5, 6, 7)) = 4.5
fn:max((2, 3, 4, 5, 6, 7)) = 7
fn:min((2, 3, 4, 5, 6, 7)) = 2
fn:sum((2, 3, 4, 5, 6, 7)) = 27
```

Conditional Expressions

```
fn:avg(
  for $r in //rcp:ingredient return
    if ( $r/@unit = "cup" )
      then xs:double($r/@amount) * 237
    else if ( $r/@unit = "teaspoon" )
      then xs:double($r/@amount) * 5
    else if ( $r/@unit = "tablespoon" )
      then xs:double($r/@amount) * 15
    else ()
)
```

Acknowledgement

- Thanks to Anders Møller, co-author of *An Introduction to XML and Web Technologies* for allowing me to use his slides **without** forcing students to buy his book!
- But if you want an in-depth XML book, the book is recommended.
  - Now also in Italian!