

## The XPath Language

Anders Møller & Michael I. Schwartzbach  
© 2006 Addison-Wesley

## Objectives

---

- Location steps and paths
- Typical locations paths
- Abbreviations
- General expressions

## XPath Expressions

---

- Flexible notation for **navigating** around trees
- A basic technology that is **widely used**
  - *uniqueness* and *scope* in XML Schema
  - *pattern matching* and *selection* in XSLT
  - *relations* in XLink and XPointer
  - *computations* on values in XSLT and XQuery

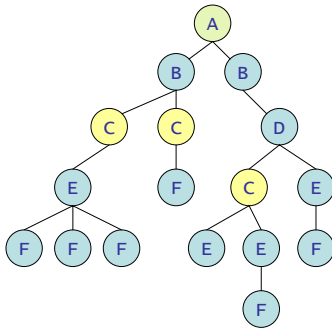
## Location Paths

---

- A *location path* evaluates to a **sequence** of nodes
- The sequence is **sorted** in document order
- The sequence will **never** contain **duplicates**

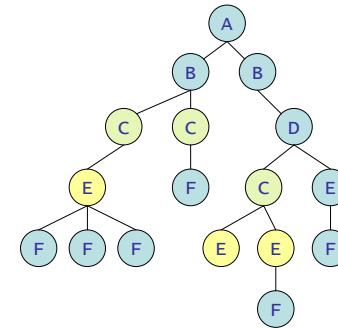


## An Example



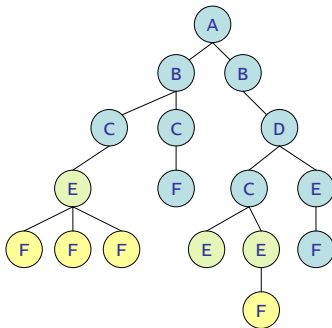
`descendant::C`

## An Example



`descendant::C/child::E`

## An Example



`descendant::C/child::E/child::F`

## Contexts

- The *context* of an XPath evaluation consists of
  - a context *node* (a node in an XML tree)
  - a context *position* and *size* (two nonnegative integers)
  - a set of *variable bindings*
  - a *function library*
  - a set of *namespace declarations*
- The application determines the initial context
- If the path starts with '/' then
  - the initial context node is the root
  - the initial position and size are 1

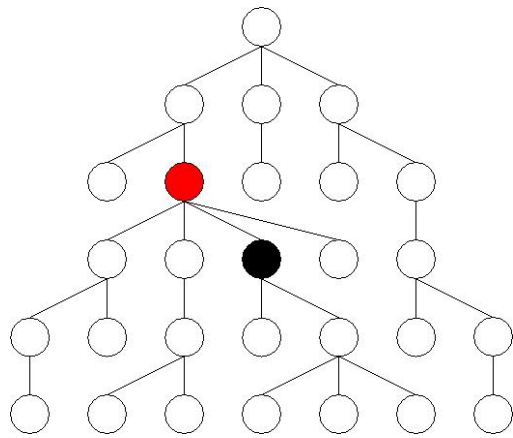
## Axes

- An axis is a sequence of nodes
- The axis is evaluated relative to the context node
- XPath supports 12 different axes
  - child
  - descendant
  - parent
  - ancestor
  - following-sibling
  - preceding-sibling
  - attribute
  - following
  - preceding
  - self
  - descendant-or-self
  - ancestor-or-self

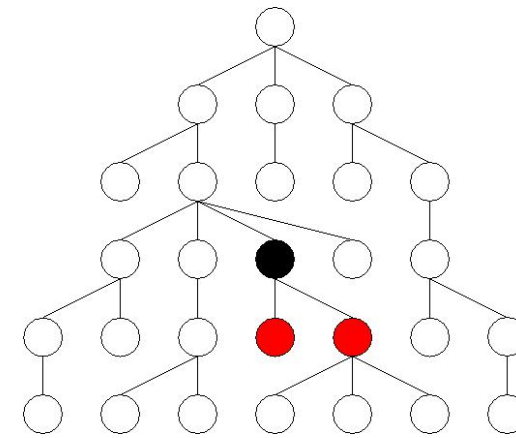
## Axis Directions

- Each axis has a *direction*
- *Forwards* means document order:
  - child, descendant, following-sibling, following, self, descendant-or-self
- *Backwards* means reverse document order:
  - parent, ancestor, preceding-sibling, preceding
- Stable but depends on the implementation:
  - attribute

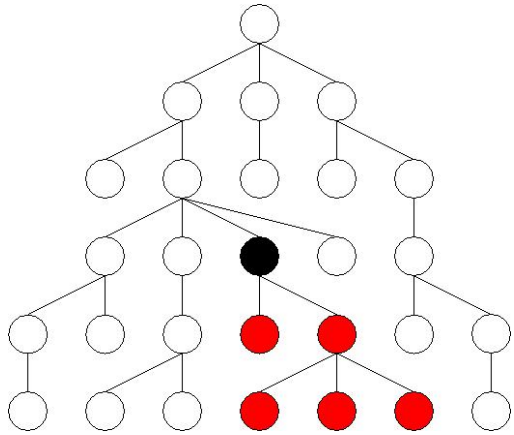
## The parent Axis



## The child Axis



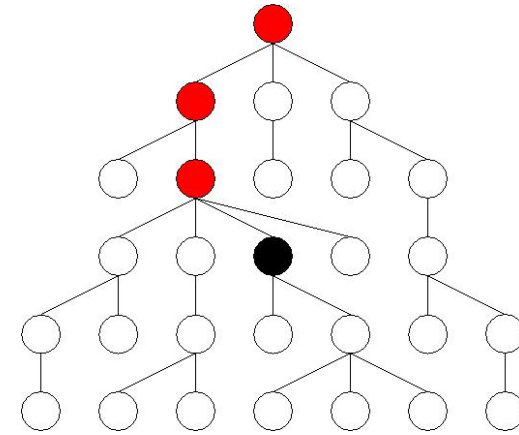
## The descendant Axis



An Introduction to XML and Web Technologies

17

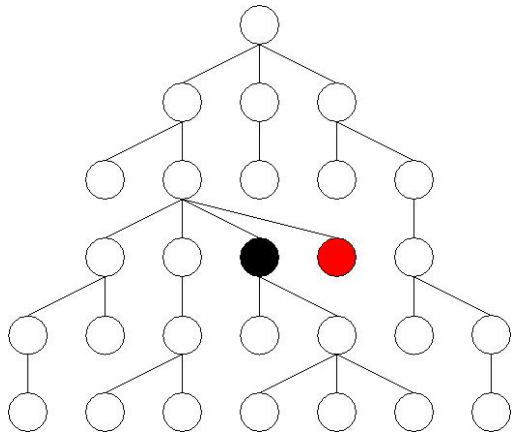
## The ancestor Axis



An Introduction to XML and Web Technologies

18

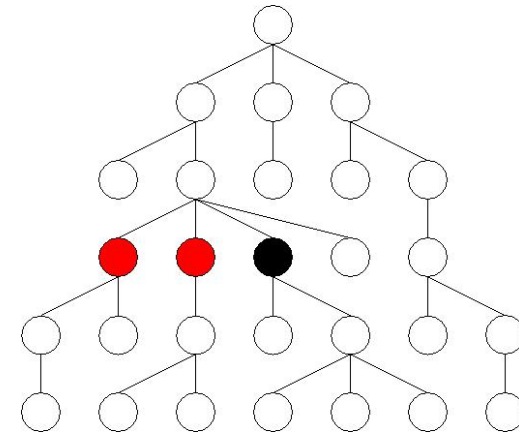
## The following-sibling Axis



An Introduction to XML and Web Technologies

19

## The preceding-sibling Axis

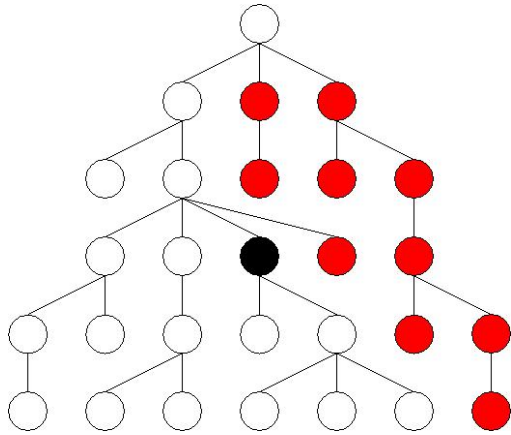


An Introduction to XML and Web Technologies

20

## The following Axis

---

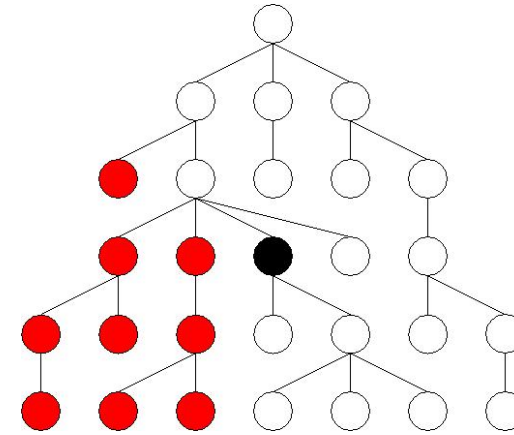


An Introduction to XML and Web Technologies

21

## The preceding Axis

---



An Introduction to XML and Web Technologies

22

## Node Tests

---

- `text()`
- `comment()`
- `processing-instruction()`
- `node()`
- `*`
- *QName*
- *\*:NCName*
- *NCName:\**

An Introduction to XML and Web Technologies

23

## Predicates

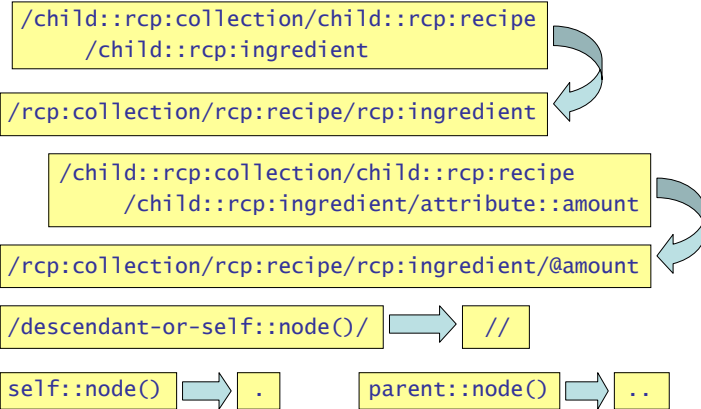
---

- General XPath expressions
- 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

An Introduction to XML and Web Technologies

24

## Abbreviations



## General Expressions

- Every expression evaluates to a *sequence* of
  - *atomic values*
  - *nodes*
- Atomic values may be
  - *numbers*
  - *booleans*
  - Unicode *strings*
  - *datatypes* defined in XML Schema
- 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*

## Literal Expressions

```
42
3.1415
6.022E23
'XPath is a lot of fun'
"XPath is a lot of fun"
'The cat said "Meow!"'
"The cat said ""Meow!"""
'XPath is just
  so much fun'
```

## Arithmetic Expressions

- +, -, \*, div, idiv, mod
- Operators are generalized to sequences
  - if any argument is empty, the result is empty
  - if all argument are singleton sequences of numbers, the operation is performed
  - otherwise, a runtime error occurs

## Variable References

```
$foo  
$bar:foo
```

- \$foo-17 refers to the variable "foo-17"
- Possible fixes:  
(\$foo)-17, \$foo -17, \$foo+-17

## Sequence Expressions

- The ',' operator concatenates sequences
- Integer ranges are constructed with 'to'
- Operators: union, intersect, except
- Sequences are always *flattened*
- These expression 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
```

## Path Expressions

- Locations paths are expressions
- They may start from arbitrary sequences
  - evaluate the path for each node
  - use the given node as context node
  - context position and size are taken from the sequence
  - the results are combined in document order

## Filter Expressions

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

```
(10 to 40)[. mod 5 = 0 and position()>20]
```

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"
```

## Node Comparison

- Operators: is, <<, >>
- Used to compare nodes on identity and order
- When applied to arbitrary values:
  - if either argument is empty, the result is empty
  - if both are singleton nodes, the nodes are compared
  - otherwise, a runtime error

```
(//rcp:recipe)[2] is  
  //rcp:recipe[rcp:title eq "Ricotta Pie"]  
/rcp:collection << (//rcp:recipe)[4]  
(//rcp:recipe)[4] >> (//rcp:recipe)[3]
```

## Be Careful About Comparisons

```
((//rcp:ingredient)[40]/@name, //rcp:ingredient)[40]/@amount) eq  
((//rcp:ingredient)[53]/@name, //rcp:ingredient)[53]/@amount)
```

Yields false, since the arguments are not singletons

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

Yields true, since the two names are found to be equal

```
((//rcp:ingredient)[40]/@name, //rcp:ingredient)[40]/@amount) is  
((//rcp:ingredient)[53]/@name, //rcp:ingredient)[53]/@amount)
```

Yields a runtime error, since the arguments are not singletons

## Algebraic Axioms for Comparisons

- Reflexivity:  $x = x$
- Symmetry:  $x = y \Rightarrow y = x$
- Transitivity:  $x = y \wedge y = z \Rightarrow x = z$   
 $x < y \wedge y < z \Rightarrow x < z$
- Anti-symmetry:  $x \leq y \wedge y \leq x \Rightarrow x = y$
- Negation:  $x \neq y \Leftrightarrow \neg(x = y)$

## XPath Violates Most Axioms

- Reflexivity?  
 $() = ()$  yields false
- Transitivity?  
 $(1, 2) = (2, 3)$ ,  $(2, 3) = (3, 4)$ , not  $(1, 2) = (3, 4)$
- Anti-symmetry?  
 $(1, 4) \leq (2, 3)$ ,  $(2, 3) \leq (1, 4)$ , not  $(1, 2) = (3, 4)$
- Negation?  
 $(1) \neq ()$  yields false,  $(1) = ()$  yields false

## Boolean Expressions

- Operators: and, or
- Arguments are coerced, false if the value is:
  - the boolean false
  - the empty sequence
  - the empty string
  - the number zero
- Constants use functions `true()` and `false()`
- Negation uses `not(...)`

## Functions

---

- XPath has an extensive *function library*
- Default *namespace* for functions:  
`http://www.w3.org/2006/xpath-functions`
- 106 functions are required
- More functions with the *namespace*:  
`http://www.w3.org/2001/XMLSchema`

## Function Invocation

---

- Calling a function with 4 arguments:  
`fn:avg(1,2,3,4)`
- Calling a function with 1 argument:  
`fn:avg((1,2,3,4))`

## Arithmetic 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
```

## Boolean Functions

---

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

## String 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"
```

## Regex Functions

```
fn:contains("XML book","XML") = fn:true()  
fn:matches("XML book","XM..[a-z]*") = fn:true()  
fn:matches("XML book",".*z.*") = fn:false()  
fn:replace("XML book","XML","web") = "web book"  
fn:replace("XML book","[a-z]","8") = "XML 8888"
```

## Cardinality Functions

```
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
```

## Sequence Functions

```
fn:distinct-values((1, 2, 3, 4, 3, 2)) = (1, 2, 3, 4)  
fn:insert-before((2, 4, 6, 8), 2, (3, 5))  
    = (2, 3, 5, 4, 6, 8)  
fn:remove((2, 4, 6, 8), 3) = (2, 4, 8)  
fn:reverse((2, 4, 6, 8)) = (8, 6, 4, 2)  
fn:subsequence((2, 4, 6, 8, 10), 2) = (4, 6, 8, 10)  
fn:subsequence((2, 4, 6, 8, 10), 2, 3) = (4, 6, 8)
```

## Aggregate Functions

```
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
```

## Node Functions

```
fn:doc("http://www.brics.dk/ixwt/recipes/recipes.xml")
fn:position()
fn:last()
```

## Coercion Functions

```
xs:integer("5") = 5
xs:integer(7.0) = 7
xs:decimal(5) = 5.0
xs:decimal("4.3") = 4.3
xs:decimal("4") = 4.0
xs:double(2) = 2.0E0
xs:double(14.3) = 1.43E1
xs:boolean(0) = fn:false()
xs:boolean("true") = fn:true()
xs:string(17) = "17"
xs:string(1.43E1) = "14.3"
xs:string(fn:true()) = "true"
```

## For Expressions

- The expression

```
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

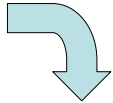
```
1, 1, 2, 1, 2, 3, 1, 2, 3, 4, 1, 2, 3, 4, 5
```

## 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 ()  
)
```

## Quantified Expressions

```
some $r in //rcp:ingredient  
  satisfies $r/@name eq "sugar"
```



```
fn:exists(  
  for $r in //rcp:ingredient return  
    if ($r/@name eq "sugar") then fn:true() else ()  
)
```

## XPath 1.0 Restrictions

- Many implementations only support XPath 1.0
- Smaller function library
- Implicit casts of values
- Some expressions change semantics:

```
"4" < "4.0"
```

is false in XPath 1.0 but true in XPath 2.0

## XPointer

- A fragment identifier mechanism based on XPath
- Different ways of pointer to the fourth recipe:

```
...#xpointer(//recipe[4])  
...#xpointer(//rcp:recipe[./rcp:title='Zuppa Inglese'])  
...#element(/1/5)  
...#r102
```

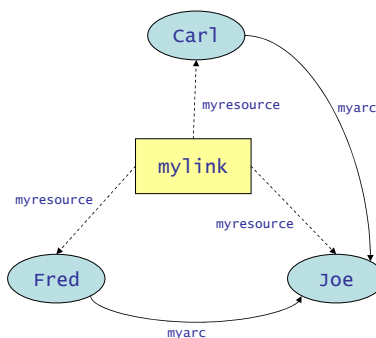
## XLink

- *Generalizing* hyperlinks from HTML to XML
- Allow *many-to-many* relations
- Allow *third party links*
- Allow *arbitrary* element names
  
- Not widely used...

## An XLink Link

```
<mylink xmlns:xlink="http://www.w3.org/1999/xlink"
  xlink:type="extended">
  <myresource xlink:type="locator"
    xlink:href="students.xml#Carl" xlink:label="student"/>
  <myresource xlink:type="locator"
    xlink:href="students.xml#Fred" xlink:label="student"/>
  <myresource xlink:type="locator"
    xlink:href="teachers.xml#Joe" xlink:label="teacher"/>
  <myarc xlink:type="arc"
    xlink:from="student" xlink:to="teacher"/>
</mylink>
```

## A Picture of the XLink



## Simple Links

```
<mylink xlink:type="simple" xlink:href="..." xlink:show="..." .../>
```



```
<mylink xlink:type="extended">
  <myresource xlink:type="resource"
    xlink:label="local"/>
  <myresource xlink:type="locator"
    xlink:label="remote" xlink:href="..."/>
  <myarc xlink:type="arc"
    xlink:from="local" xlink:to="remote" xlink:show="..."
  .../>
</mylink>
```

## HTML Links in XLink

---

```
<a xlink:type="simple"
  xlink:href="..."
  xlink:show="replace"
  xlink:actuate="onRequest"/>
```

## The HLink Alternative

---

```
<hlink namespace="http://www.w3.org/1999/xhtml"
  element="a"
  locator="@href"
  effect="replace"
  actuate="onRequest"
  replacement="@target"/>
```

## Essential Online Resources

---

- <http://www.w3.org/TR/xpath/>
- <http://www.w3.org/TR/xpath20/>
- <http://www.w3.org/TR/xlink/>
- <http://www.w3.org/TR/xptr-framework/>