NC STATE UNIVERSITY

ENGINEERING ONLINE

Lecture Notes

Course Number:	CSC 513
Instructor:	Dr. Singh
Lecture Number:	23

XPath

[ ]

[1]

[last()]

[0] nothing

[-1] "

[last()+1] "

EFFECTIVE BOOLEAN VALUE

$[\underbrace{\quad\sim\sim\quad}]$ if true() then produce answer
false() omit answer

$\underline{p}$ or $\underline{q}$

p and q

0 ↦ false

$[\underbrace{\text{<elem/>}}_{\text{true}}]$

GOTCHA

[-1 or -1]

XQuery

- The official query language for XML, now a W3C recommendation, as version 1.0
- Given a non-XML syntax, easier on the human eye than XML
- An XML rendition, **XqueryX**, is in the works

XQuery Basic Paradigm

A variable

The basic paradigm mimics the SQL (SELECT–FROM–WHERE) clause

```
for $x in doc('q2.xml')//Song
where $x/@lg = 'en'
return
 <English-Sgr name='{$x/Sgr/@name}' ti='{$x/@ti}'/>
```

node sequence

string

construct an element

attribute

values

signifies a value to be computed

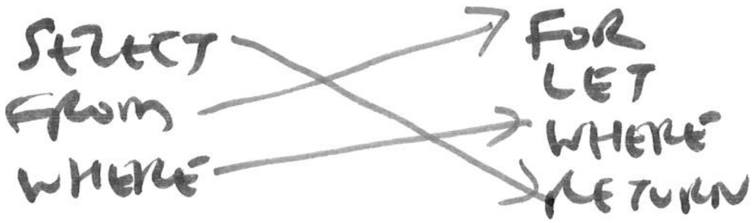
FLWOR Expressions

Pronounced "flower"

- For: iterative binding of variables over range of values
- Let: one shot binding of variables over vector of values
- Where (optional)
- Order by (sort: optional)
- Return (required)

Need at least one of **for** or **let**

XQuery For Clause



The **for** clause

- Introduces one or more variables ≈ tuple variables in SQL
- Generates possible bindings for each variable
- Acts as a mapping functor or iterator
  - In essence, all possible combinations of bindings are generated: like a Cartesian product in relational algebra
  - The bindings form an ordered list

XQuery Where Clause

The **where** clause

- Selects the combinations of bindings that are desired
- Behaves like the **where** clause in SQL, in essence producing a join based on the Cartesian product

XQuery Return Clause

The **return** clause

- Specifies what node-sets are returned based on the selected combinations of bindings

XQuery Let Clause

The **let** clause

- Like **for**, introduces one or more variables

not a tuple
but a relation
(sequence of tuples)

- Like **for**, generates possible bindings for each variable
- Unlike **for**, generates the bindings as a list in one shot (no iteration)

XQuery Order By Clause

The **order by** clause

- Specifies how the vector of variable bindings is to be sorted before the return clause
- Sorting expressions can be nested by separating them with commas
- Variants allow specifying
  - **descending** or **ascending** (default)
  - **empty greatest** or **empty least** to accommodate empty elements
  - stable sorts: **stable order by**
  - collations: **order by $t collation** collation-URI: (obscure, so skip)

not emphasized

XQuery Positional Variables

The **for** clause can be enhanced with a positional variable

- A positional variable captures the position of the main variable in the given **for** clause with respect to the expression from which the main variable is generated
- Introduce a positional variable via the **at** $var construct

```
let $y := ....
for $x in $y at $i

return ...$i
```

```
for $x in doc . . . //item ~~at~~ at $pos
for $y in doc . . . //item
```

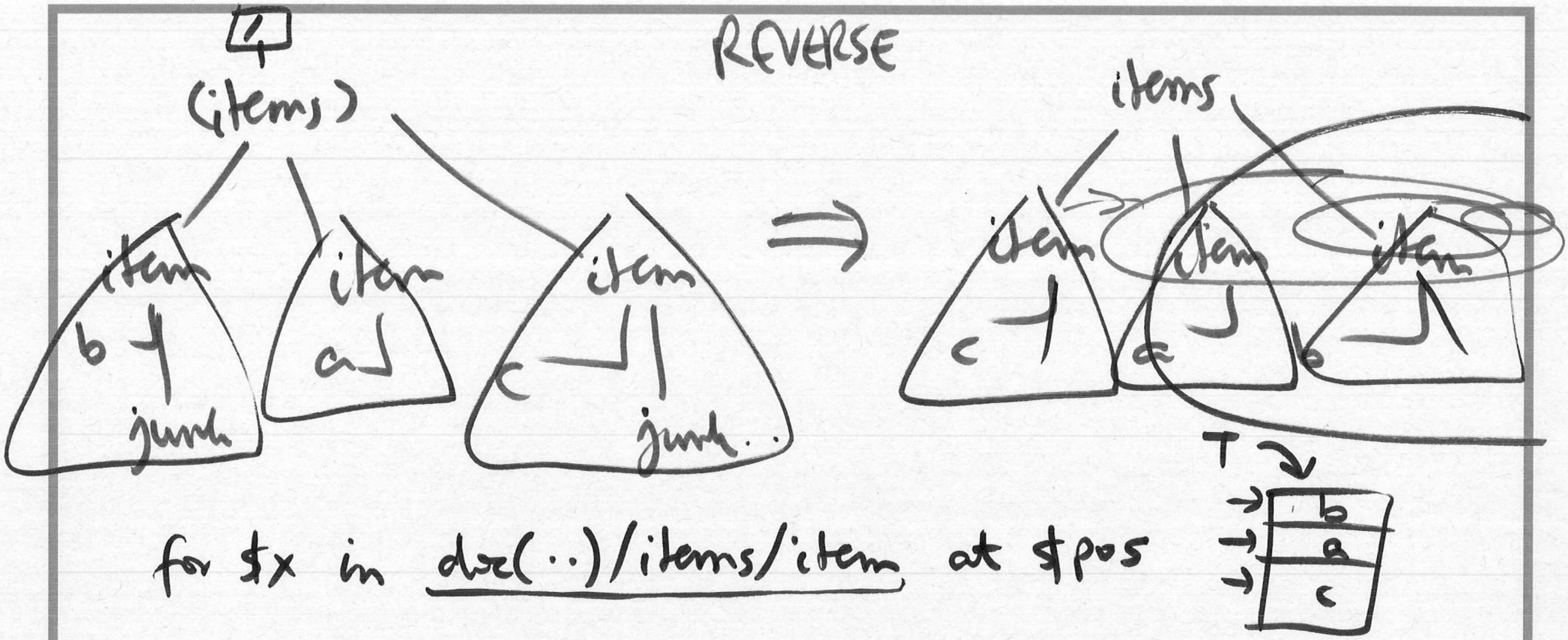
```
? let $all := doc . . //item        (b, a, c)
                                    ^ array or table
                                    sequence
```

```
for $x in . . .          at $pos
order by descending
            $pos
return $x
```

```
for $x in . . .
```

```
let $all := . . .
for $x in $all at $pos
→ same
```

REVERSE



```
for $x in doc(..)/items/item at $pos

return   $x[last() - $pos+1]

         b[last() - ..]
         a[           ]
         c[           ]
```

T[last()] = c

following-sibling

function ≈ method

```
function next-sib ($x) {
  $x/following-sibling::element()[1]
```

→<elem attr='a' <elem attr='a'
abc ⟹ abc
<next/> ⇏
</elem> </elem>

name($x)='elem'

$x/@* ⟶

$x/text() ⟶

```
return ...
    element name($x)
       {
         $x/@*,
         $x/text()
       }

    $x/*
```

```
<student>                      <course>
   <course/>       =>            <student/>
</student>                     </course>
```

X
deep

```
<course>
   <student>
      <course/>
```