Universität Freiburg
Institut für Informatik
Prof. Dr. Peter Fischer
Lecuture XML and Databases
Winter Semester 2011/12

Due date/discussion: 27.1.2012

Exercise Sheet 10

XQuery Implementation and Optimisation

Exercise 1: Query plans: Implementation

The purpose of this exercise and of the following exercise is to familiarise you with query plans and optimisation of query plans for XQuery. At the end of this exercise sheet, you will find a short introduction to query plans.

Draw the query plans for the following queries and check them with MXQuery.

2.1. (From exercise sheet 7, exercise 1.2.)

```
let $doc := doc("flights.xml")
let $results := (
<order>
  {
  for $a in $doc//Airport
  let $c := count($doc//Flight[((./source eq $a/@airId)
                              or
                              (./destination eq $a/@airId))
                             ])
  order by $c descending, $a/name ascending
  return
  <result>
    {$a}
    <count>{$c}</count>
  </result>
  }
</order>
)
let $maxcount := xs:integer(max($results//count))
return $results/result[xs:integer(./count) eq $maxcount]
```

2.2. (From exercise sheet 7, exercise 1.4.)

```
let $doc := doc("flights.xml")
return
<Possibilities>{
  for $f1 in $doc//Flight[./source eq 'NPL'],
      $f3 in $doc//Flight[destination eq 'SPL'],
      $f2 in $doc//Flight
  where $f1/destination eq $f2/source
        and $f2/destination eq $f3/source
        and xs:time($f1/arrival) lt xs:time($f2/departure)
        and xs:time($f2/arrival) lt xs:time($f3/departure)
        and $f1/seats > 0 and $f2/seats > 0 and $f3/seats > 0
  return
  <TwoIntermediateStops>
```

```
    {$f1}
    {$f2}
    {$f3}
  </TwoIntermediateStops>
}</Possibilities>
```

2.3. (From exercise sheet 6, exercise 1.3.)

```
for $a in distinct-values(doc("bib.xml")//author)
return <res>
<name>{$a}</name>
<count>
{
count(doc("bib.xml")//book[author = $a])
}
</count>
</res>
```

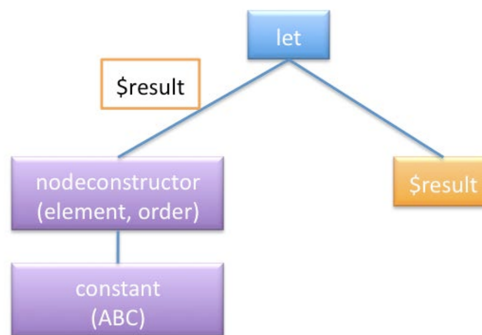
Exercise 2 - Optimisation

2.1. Estimate the execution cost of the query 2.3. (find a suitable measure for expressing the cost, e.g., number of nodes in the query plan which are used during execution)
2.2. Optimise the query plan of this query and show, using the chosen metric, that the rewritten plan is better.

Introduction to query plans

Let/return (a return belongs to a let or for)

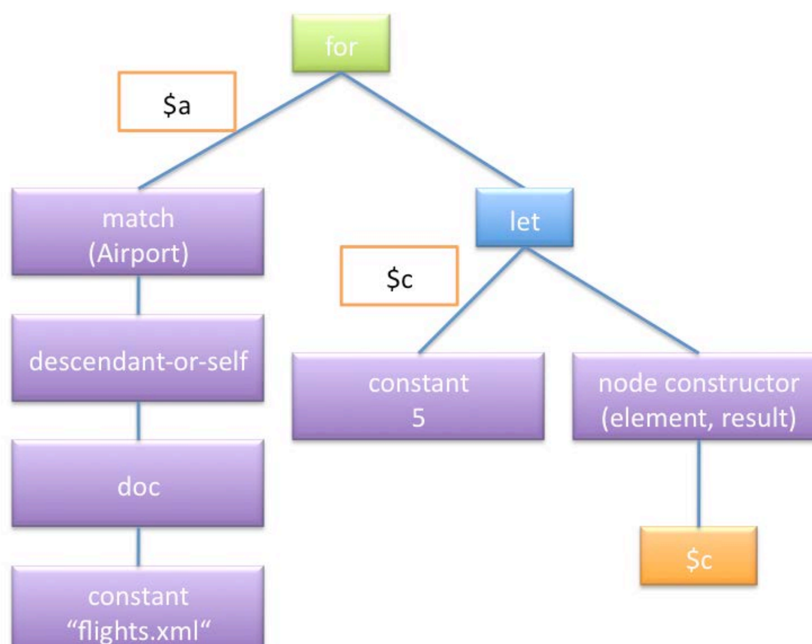
```
let $result := <order>ABC</order>
return $result
```



The let variable is represented as a label on the left edge; the right edge (corresponding to the initial return) gives the result and cannot be executed before the left edge.

For/return

```
for $a in doc("flights.xml")//Airport
let $c := 5
return $result <result>{$c}</result>
```



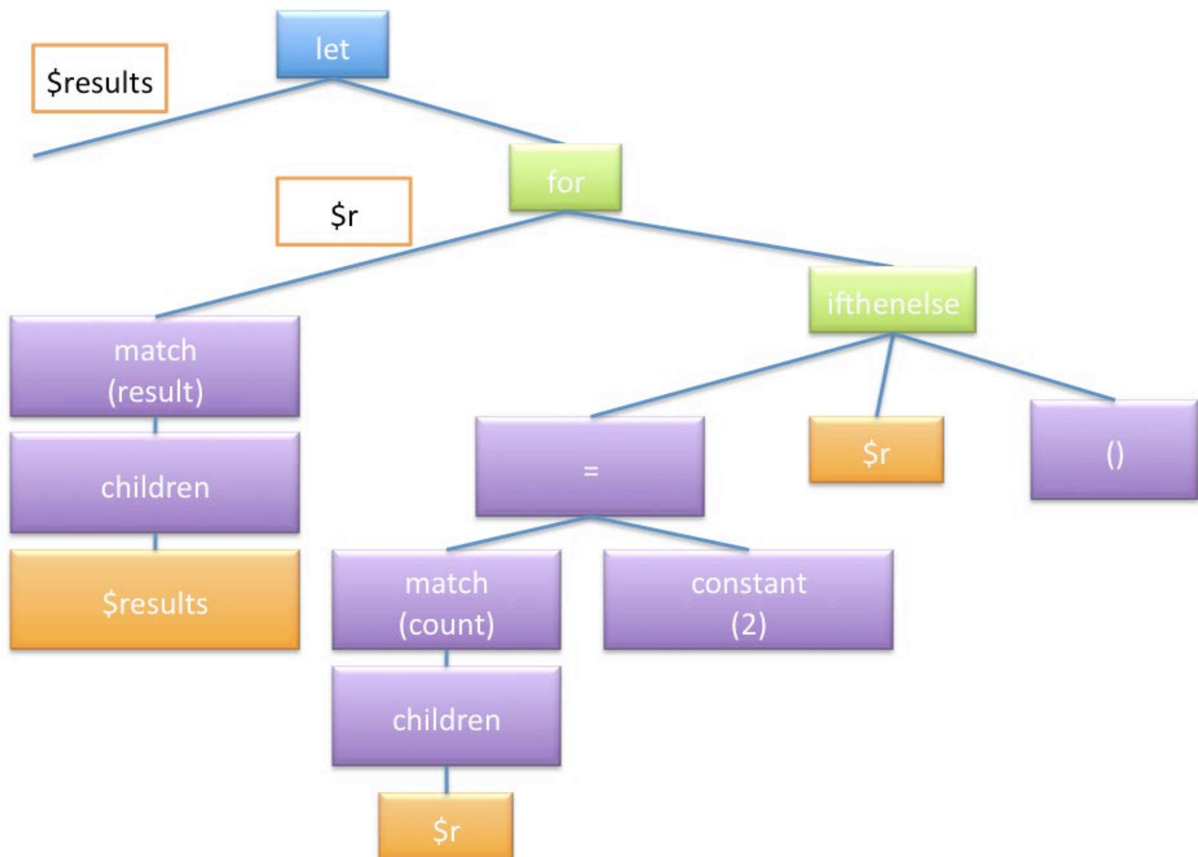
The for variable is represented as a label on the left edge, which evaluates to a sequence of nodes. For each mapping of the variable to one of the items in the sequence generated by the left edge, the right edge is executed.
The result of the for is the sequence of all results returned on the right side for each mapping.
A branch is to be read bottom-up: the "flights.xml" is used, then the doc is applied to it, then we evaluate descendant-or-self on this node, get the Airport, and the result is bound to $a.

If-then-else

```
let $results := [...]
return $results/result[count = 2]
```

equivalent to

```
let $results := [...]
for $r in $results/result
where count = 2
return $r
```



A temporary variable $r is introduced. The condition is expressed using an ifthenelse node. It takes a boolean expression in the first branch, returns the result of the second

branch when the expression is true, and the result of the third branch when the expression is false (in our case, the empty sequence).

You can visualise the query plan by invoking MXQuery as follows:

java -jar mxquery.jar -f queryfile.xq --explain

The expression tree is written in an XML form, which you can copy and paste into an XML editor like Oxygen to display the structure of the query tree.

Note that the MXQuery engine does some optimisations, hides some details and compacts the query plan, so that the notation here may differ from that of MXQuery, for example:

- $variable_name becomes VariableIterator("variable_name")
- $results - children - match("result") becomes ChildrenIterator(VariableIterator("results"), "result")
- $doc - descendant-or-self - match ("Flight") becomes DescendantOrSelfIterator(VariableIterator("doc"), "Flight")
- Inside a for loop, ifthenelse can be a Where iterator
- A function call is represented as a FunctionCallIterator
- A node constructor is represented as XMLContent.