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

Due date/discussion: 27.1.2012

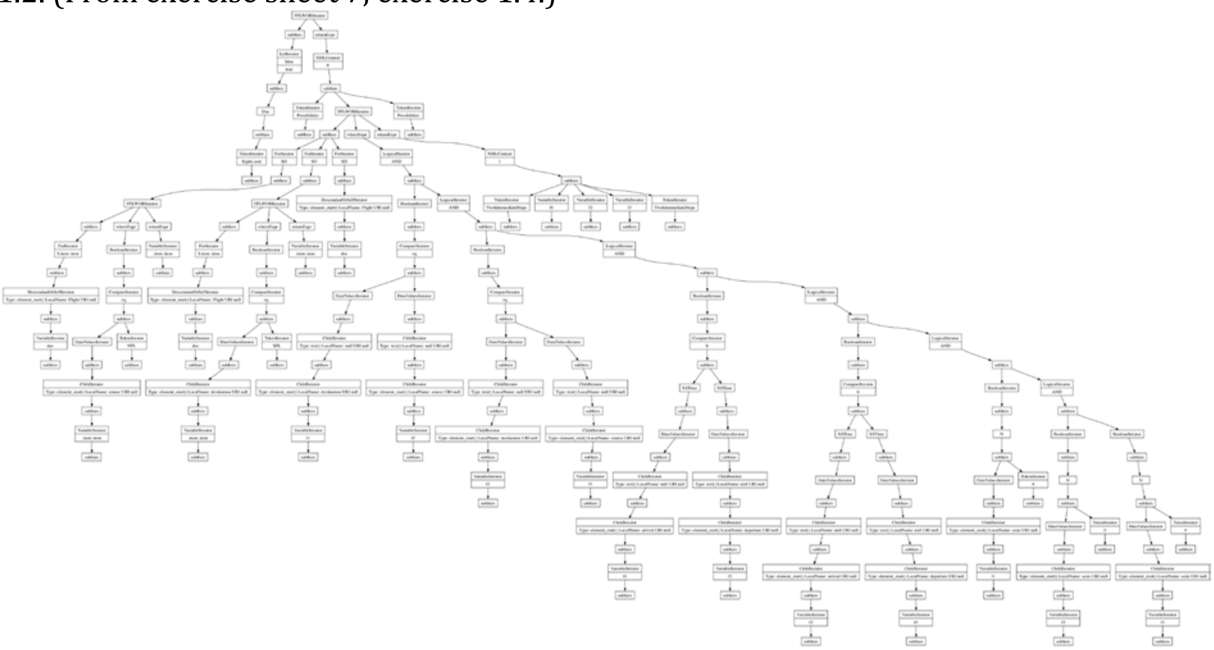
Solution Sheet 10

XQuery Implementation and Optimisation

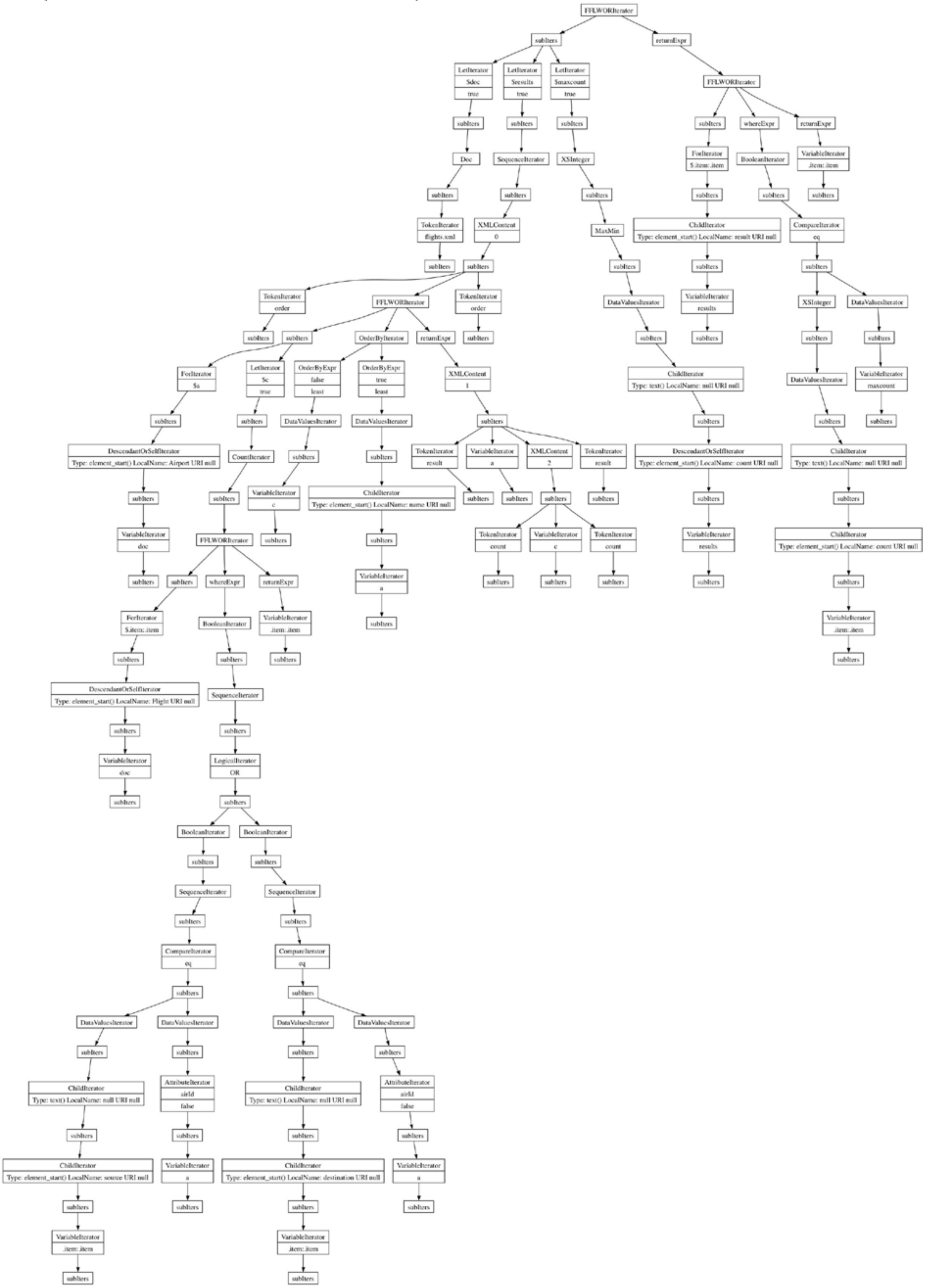
Exercise 1: Query plans: Implementation

The trees look as follows:

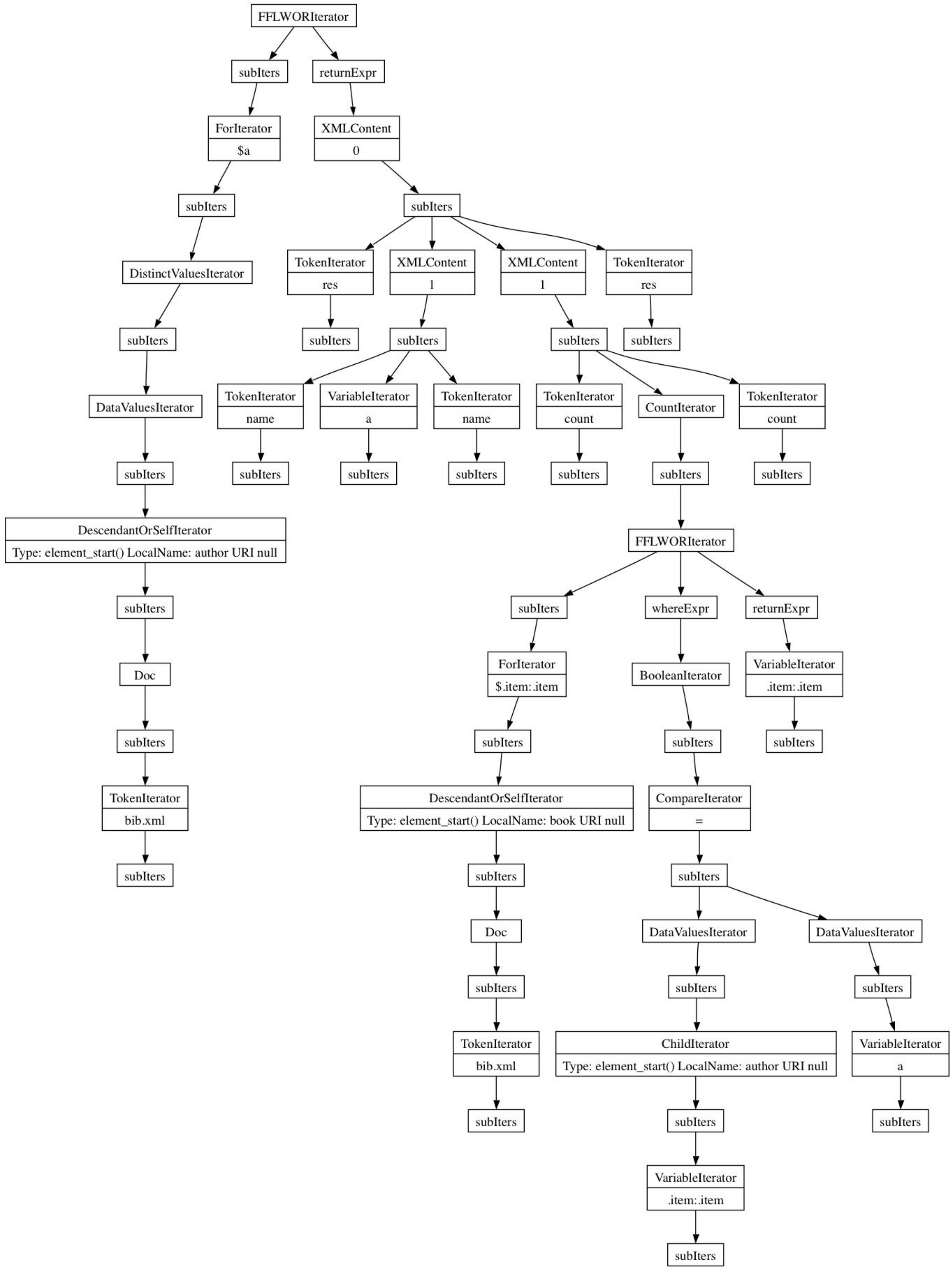
1.2. (From exercise sheet 7, exercise 1.4.)



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

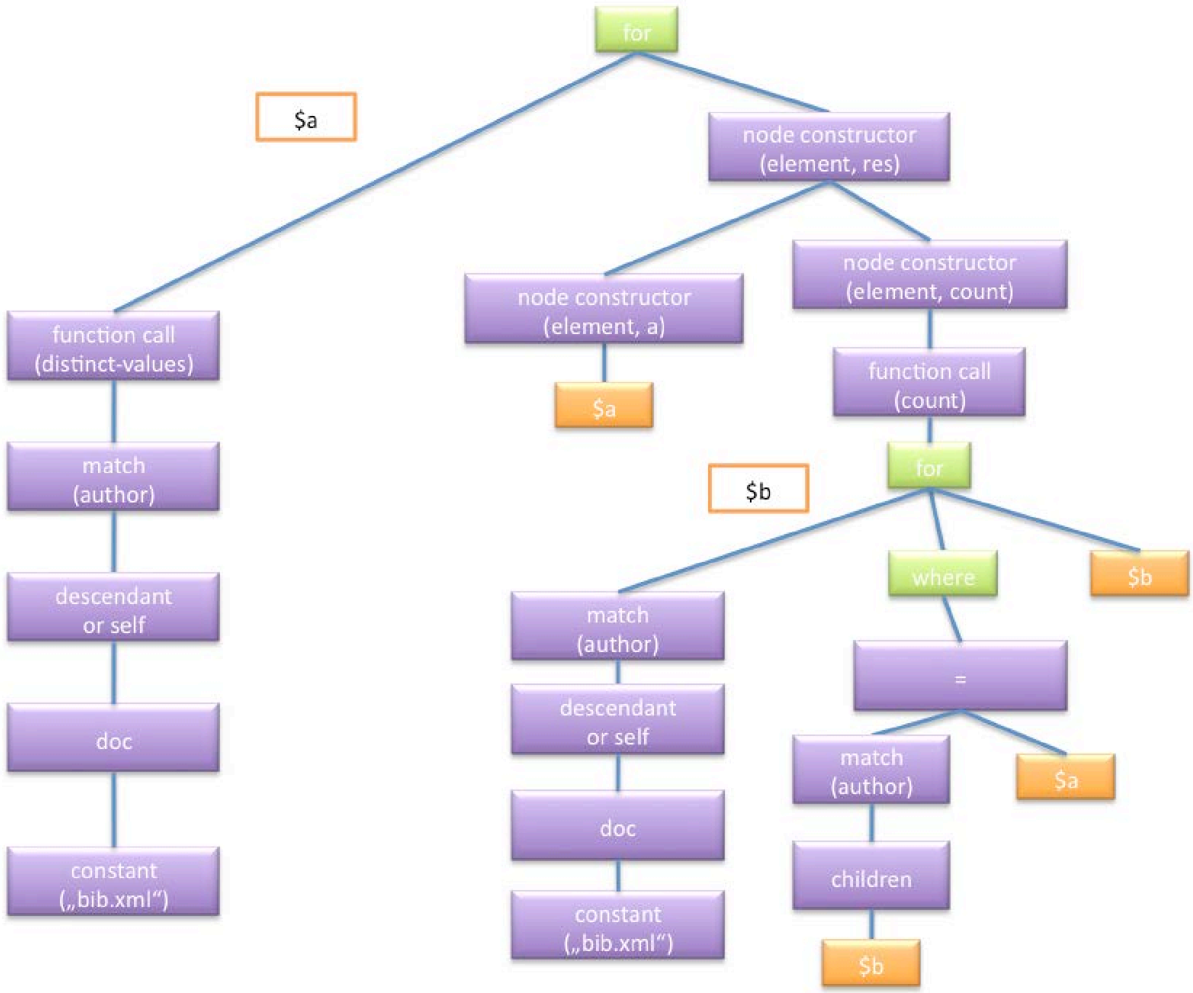


1.3. (From exercise sheet 6, exercise 1.3.)



Exercise 2 - Optimisation

Here is a more user-friendly version of the tree:

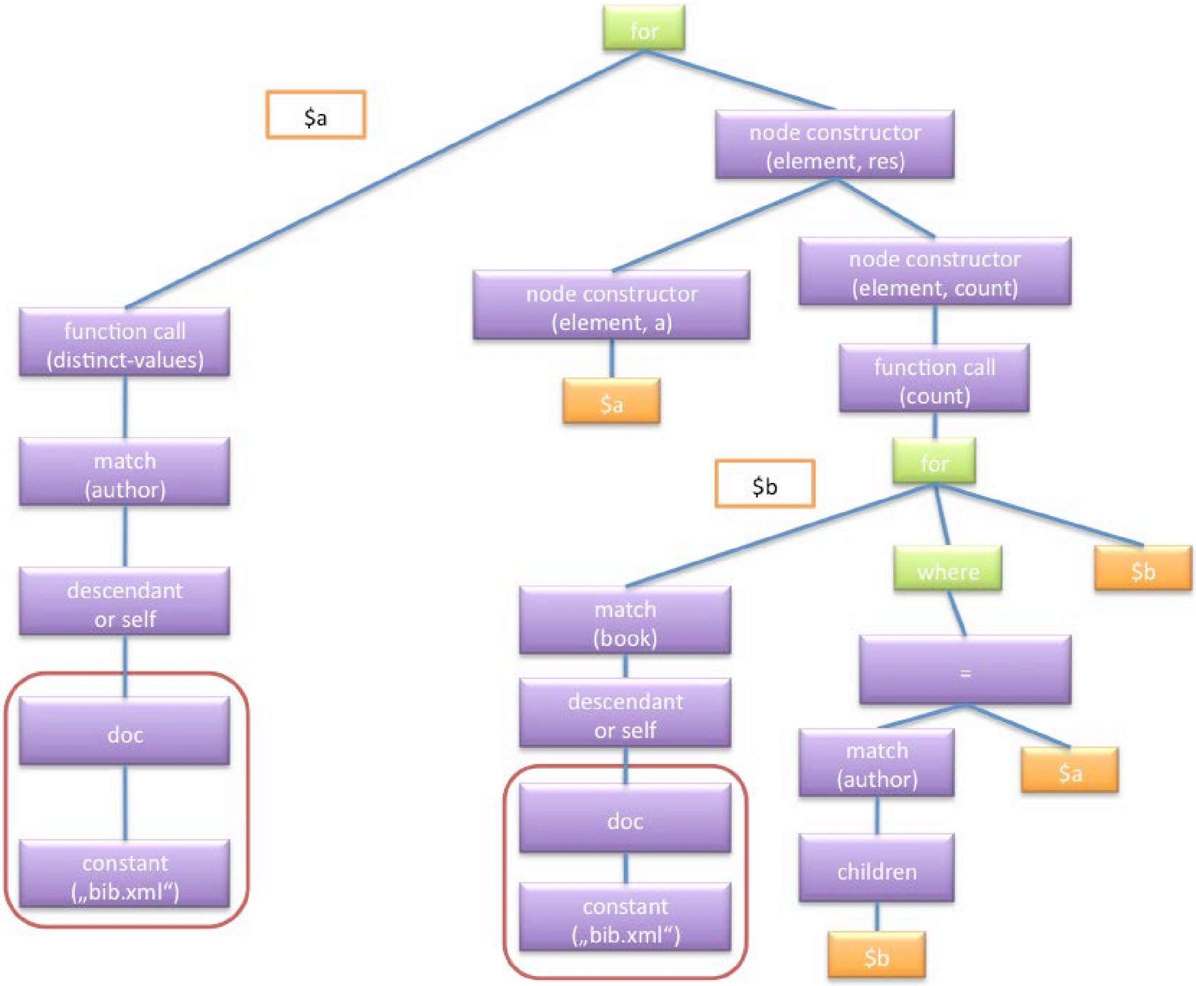


We define our cost function as the number of nodes which are read/used during execution.

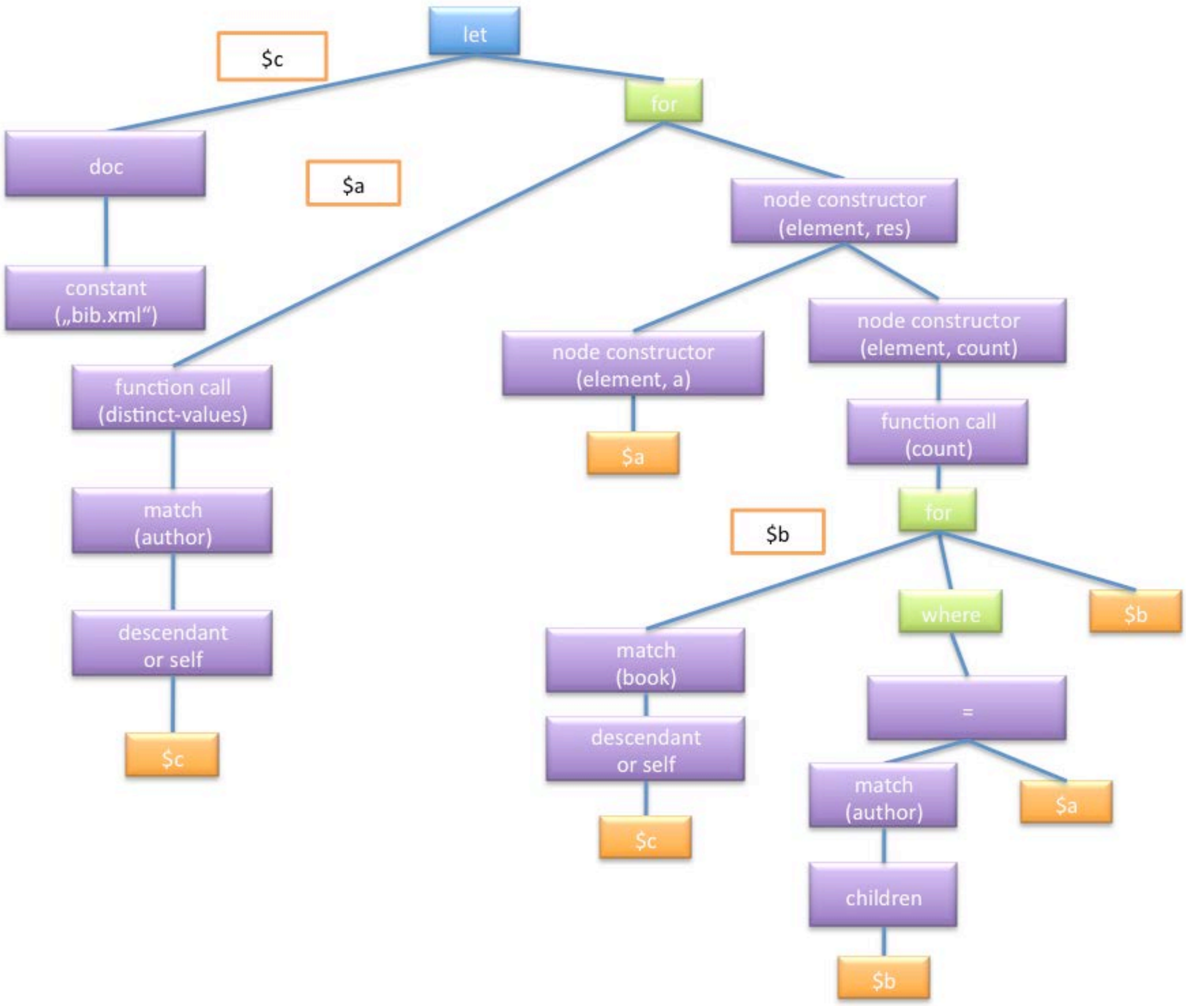
The current cost is

C1 = 1 + 5 + A(5+1+4+B*7) = 6 + 10A + 7AB

3.2. Generally speaking, optimising is about identifying operations which can be removed (not necessary) or optimised(i.e., executed only once). This is for example the case for common subexpressions:



For such an expression, we can introduce a let statement and a temporary variable, which will be executed once for each context where it is executed several times (i.e. here, outside the outermost for loop). We then use this temporary variable wherever this common subexpression was used.



This is equivalent to the following query:

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

The new cost is

$C2 = 3+1+4+A(5+1+3+B(7)) = 8 + 9A+7AB$

This is better when C1>C2 , i.e., as soon as A>2, which is the case in practice.

In a broader scope, the query could be optimized by

- Using Indexes on the inner loop
- Recognizing the GROUP BY semantics, and rewriting it to use an explicit GROUP BY operator which could be based on sorting or hashing