

# Pruning Nested XQuery Queries

Bilel Gueni

Talel Abdessalem, Bogdan Cautis, Emmanuel Waller

TELECOM ParisTech - Université de Paris-Sud

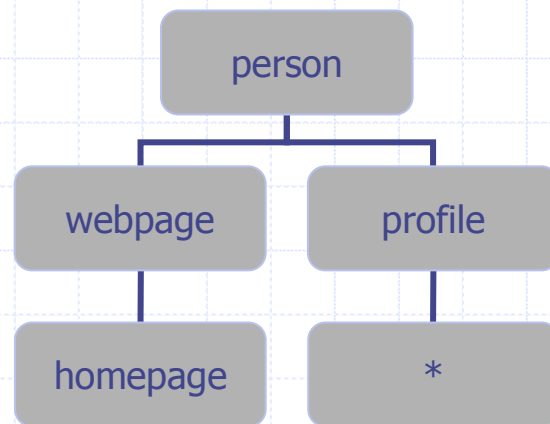
FRANCE

# Introduction

- XQuery optimization is crucial and complex
  - Complex : operational semantics
  - Crucial : evaluation efficiency
    - Handling many document accesses is expensive
    - Construction of XML elements is expensive
- Challenge
  - Minimizing document access and intermediary element construction
- Various approaches
  - Projecting XML documents
  - Type based projections
  - Reformulating XQuery queries
  - ...
- Our contribution
  - Rewriting rules for nested XQuery queries **with composition**

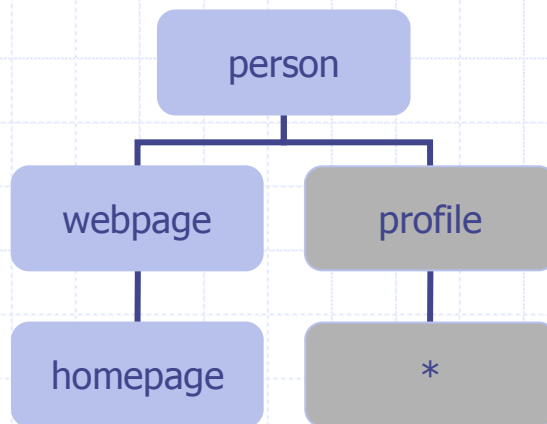
# Example of query composition

```
for $j in doc("auction.xml")/site/people/person
return <person>
  <webpage>{$j/homepage}</webpage>
  <profile>{$j/profile/*}</profile>
</person>
```



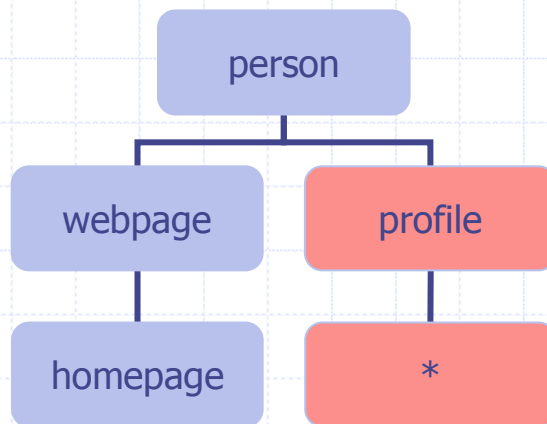
# Example of query composition

```
let $i := (for $j in doc("auction.xml")/site/people/person
return <person>
  <webpage>{$j/homepage}</webpage>
  <profile>{$j/profile/*}</profile>
</person> )
return $i/webpage
```



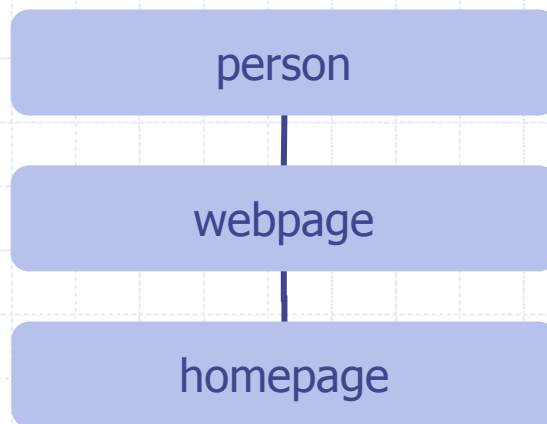
# Example of query composition

```
let $i := (for $j in doc("auction.xml")/site/people/person
return <person>
  <webpage>{$j/homepage}</webpage>
  <profile>{$j/profile/*}</profile>
</person>)
return $i/webpage
```



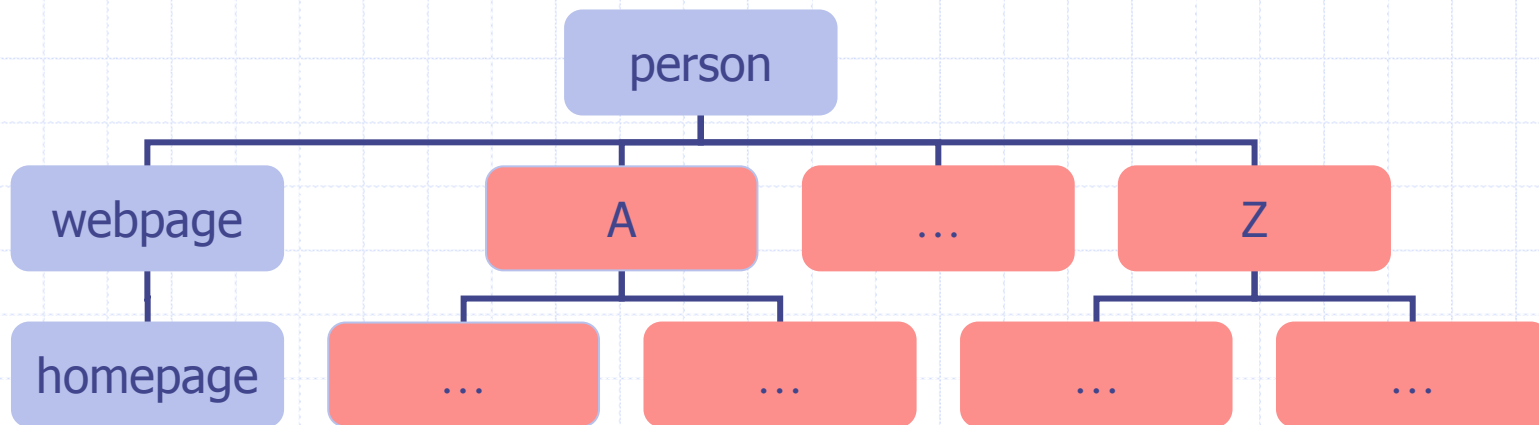
# Example of query composition

```
let $i := (for $j in doc("auction.xml")/site/people/person
return <person>
  <webpage>{$j/homepage}</webpage>
</person>)
return $i/webpage
```



# Example of query composition

```
let $i := (for $j in doc("auction.xml")/site/people/person
return <person>
  <webpage>{$j/homepage}</webpage>
A FLWR or any sub-expression
</person>)
return $i/webpage
```



# The main idea

- In the case of query composition, some intermediate results/computations may not be necessary
  - Sub-expressions generating them are useless
- Our algorithm:
  - Analyzes the input query in order to detect these sub-expressions
  - Projects out (prune) these sub-expressions
  - Maintain equivalence with the original query

# Preliminaries

- Intermediate results/computations
  - Associated to variables
- For each variable  $\$i$ 
  - its **definition** denotes the sub-expression producing the results “stored” in  $\$i$
  - its **scope** denotes the sub-expressions where  $\$i$  can be used

# Preliminaries

- Intermediate results/computations
  - Associated to variables
- For each variable  $\$i$ 
  - its **definition** denotes the sub-expression producing the results “stored” in  $\$i$
  - its **scope** denotes the sub-expressions where  $\$i$  can be used

let  $\$i :=$  *definition of  $\$i$*   
where *part of the scope of  $\$i$*   
return *part of the scope of  $\$i$*

for  $\$i$  in *definition of  $\$i$*   
where *part of the scope of  $\$i$*   
return *part of the scope of  $\$i$*

# Preliminaries

- Intermediate results/computations
  - Associated to variables
- For each variable  $\$i$ 
  - its **definition** denotes the sub-expression producing the results “stored” in  $\$i$
  - its **scope** denotes the sub-expressions where  $\$i$  can be used

let  $\$i :=$  *definition of  $\$i$*   
where *part of the scope of  $\$i$*   
return *part of the scope of  $\$i$*

for  $\$i$  in *definition of  $\$i$*   
where *part of the scope of  $\$i$*   
return *part of the scope of  $\$i$*

based on the references to  $\$i$  in the scope, simplify its definition

# Algorithm overview

A bottom-up static analysis of input query

For each variable  $\$i$ , three main steps:

1. Apply recursively the pruning process on its definition and scope
2. Extract from the scope the paths referencing  $\$i$   
Explicit references:  $\$i/A/B/\dots$   
Implicit references:  $\$k/A/B/\dots$ , while  $\$k$  is linked to  $\$i$   
(ex. for  $\$k$  in  $\$i/\dots$ , or let  $\$k:=\$i/\dots$ )
3. Using the paths from Step 2, detect and project out the unneeded parts from the definition of  $\$i$

# Path extraction

- Returned vs used paths [Marian and Simeon, VLDB'03]
  - Paths determine XML result nodes
  - In some cases (**used paths**), the subtrees rooted at these result nodes are not necessary for the query evaluation  
For example, when the result nodes are only used for iteration
- **extractPaths(\$i, scope of \$i) => P, P#**
  - Analyze the scope of \$i
  - Return the paths referencing \$i: P for used paths, and P# for returned paths

# Path extraction

- Returned vs used paths [Marian and Simeon, VLDB'03]
  - Paths determine XML result nodes
  - In some cases (**used paths**), the subtrees rooted at these result nodes are not necessary for the query evaluation
    - For example, when the result nodes are only used for iteration
- **extractPaths(\$i, scope of \$i) => P, P#**
  - Analyze the scope of \$i
  - Return the paths referencing \$i: P for used paths, and P# for returned paths
  - **Described in the paper by a set of inference rules**
    - each rule details the path extraction process according to the query structure of the input scope

# Pruning $\$i$ 's definition

$\text{projectPaths}(P, P\#, \$i\text{'s definition}) \Rightarrow \text{definition}'$

- Analyze the definition of  $\$i$  and its expected result
- Determine the result nodes that are reached by some paths in  $P$  and  $P\#$ 
  - Those that are not reached are unneeded
- Determine whether the descendent nodes are necessary or not
  - If a node is only reached by used paths, its descendents are not needed
- Project out the sub-expressions that compute the unneeded nodes (and nothing else)

# Pruning $\$i$ 's definition

$\text{projectPaths}(P, P\#, \$i\text{'s definition}) \Rightarrow \text{definition}'$

- Analyze the definition of  $\$i$  and its expected result
- Determine the result nodes that are reached by some paths in  $P$  and  $P\#$ 
  - Those that are not reached are unneeded
- Determine whether the descendent nodes are necessary or not
  - If a node is only reached by used paths, its descendents are not needed
- Project out the sub-expressions that compute the unneeded nodes (and nothing else)
- Detailed in the paper by a set of rewriting rules
  - each rule describes the path projection process according to the query structure of the input definition

# Pruning rule example

FLWR expression:

$$\text{Env} \vdash \text{prune}(e_1) \Rightarrow e'_1$$
$$\text{Env.add}(\$i)$$
$$\text{Env} \vdash \text{prune}(e_2) \Rightarrow e'_2$$
$$\text{Env} \vdash \text{extractPaths}(\$i, e'_2) \Rightarrow P, P\#$$
$$\text{Env} \vdash \text{projectPaths}(P, P\#, e'_1) \Rightarrow e''_1$$

---

$$\text{Env} \vdash \text{prune}(\text{let } \$i := e_1 \text{ return } e_2) \Rightarrow \text{let } \$i := e''_1 \text{ return } e'_2$$

# Pruning rule example

FLWR expression:

$\text{Env} \vdash \text{prune}(e_1) \Rightarrow e'_1$

$\text{Env.add}(\$i)$

$\text{Env} \vdash \text{prune}(e_2) \Rightarrow e'_2$

$\text{Env} \vdash \text{extractPaths}(\$i, e'_2) \Rightarrow P, P\#$

$\text{Env} \vdash \text{projectPaths}(P, P\#, e'_1) \Rightarrow e''_1$

---

$\text{Env} \vdash \text{prune}(\text{let } \$i := e_1 \text{ return } e_2) \Rightarrow \text{let } \$i := e''_1 \text{ return } e'_2$

# Pruning rule example

FLWR expression:

$$\text{Env} \vdash \text{prune}(e_1) \Rightarrow e'_1$$
$$\text{Env.add}(\$i)$$
$$\text{Env} \vdash \text{prune}(e_2) \Rightarrow e'_2$$
$$\text{Env} \vdash \text{extractPaths}(\$i, e'_2) \Rightarrow P, P\#$$
$$\text{Env} \vdash \text{projectPaths}(P, P\#, e'_1) \Rightarrow e''_1$$

---

$$\text{Env} \vdash \text{prune}(\text{let } \$i := e_1 \text{ return } e_2) \Rightarrow \text{let } \$i := e''_1 \text{ return } e'_2$$

# Pruning rule example

FLWR expression:

$$\text{Env} \vdash \text{prune}(e_1) \Rightarrow e'_1$$
$$\text{Env.add}(\$i)$$
$$\text{Env} \vdash \text{prune}(e_2) \Rightarrow e'_2$$
$$\text{Env} \vdash \text{extractPaths}(\$i, e'_2) \Rightarrow P, P\#$$
$$\text{Env} \vdash \text{projectPaths}(P, P\#, e'_1) \Rightarrow e''_1$$

---

$$\text{Env} \vdash \text{prune}(\text{let } \$i := e_1 \text{ return } e_2) \Rightarrow \text{let } \$i := e''_1 \text{ return } e'_2$$

# Pruning rule example

FLWR expression:

$$\text{Env} \vdash \text{prune}(e_1) \Rightarrow e'_1$$
$$\text{Env.add}(\$i)$$
$$\text{Env} \vdash \text{prune}(e_2) \Rightarrow e'_2$$
$$\text{Env} \vdash \text{extractPaths}(\$i, e'_2) \Rightarrow P, P\#$$
$$\text{Env} \vdash \text{projectPaths}(P, P\#, e'_1) \Rightarrow e''_1$$

---

$$\text{Env} \vdash \text{prune}(\text{let } \$i := e_1 \text{ return } e_2) \Rightarrow \text{let } \$i := e''_1 \text{ return } e'_2$$

# Pruning rule example

FLWR expression:

$\text{Env} \vdash \text{prune}(e_1) \Rightarrow e'_1$

$\text{Env.add}(\$i)$

$\text{Env} \vdash \text{prune}(e_2) \Rightarrow e'_2 = ()$

$\text{Env} \vdash \text{extractPaths}(\$i, e'_2) \Rightarrow P, P\#$

$\text{Env} \vdash \text{projectPaths}(P, P\#, e'_1) \Rightarrow e''_1$

---

$\text{Env} \vdash \text{prune}(\text{let } \$i := e_1 \text{ return } e_2) \Rightarrow \underbrace{\text{let } \$i := e''_1 \text{ return } e'_2}_{= ()}$

# Pruning rule example

FLWR expression:

$\text{Env} \vdash \text{prune}(e_1) \Rightarrow e'_1$

$\text{Env.add}(\$i)$

$\text{Env} \vdash \text{prune}(e_2) \Rightarrow ()$

---

$\text{Env} \vdash \text{prune}(\text{let } \$i := e_1 \text{ return } e_2) \Rightarrow ()$

# Equivalence

Theorem:

For  $q$  an XQuery expression, if  $\text{Env} \Vdash \text{prune}(q) \Rightarrow q'$   
then  $\forall I, q(I) =_{\text{deep}} q'(I)$ .

- Proof by induction on the rewrite rules
  - Details can be found in the paper
- Intuition: the computations we remove are not “linked” to the root computations

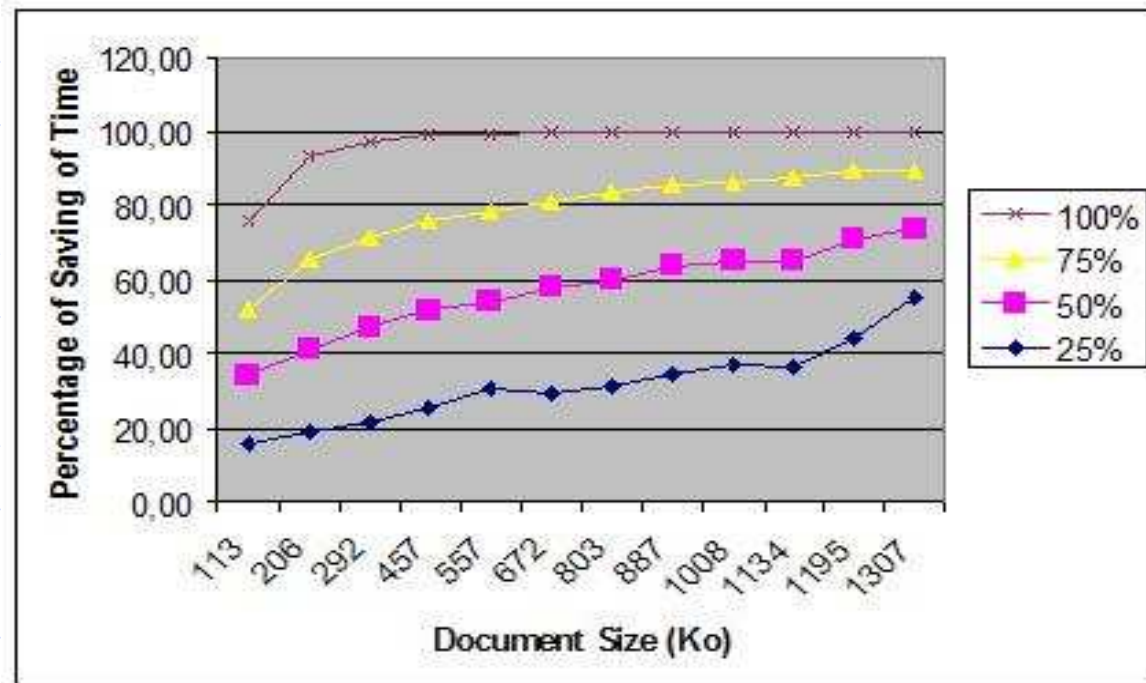
# Experiments

- We measured the impact on evaluation time
- Using a template query, we varied
  - the nature and complexity of pruned sub-expressions
    - FLWR blocks
    - explicit element construction
    - XPath expressions
  - the percentage of useless intermediate results
  - the size of the source document

# Experiments

```
let $i := <personInf>
  {for $j in doc("xmark.xml")/site/people/person
   return (<name>{test_expr}</name>, <age>{test_expr}</age>,
          <gender>{test_expr}</gender>, <email>{test_expr}</email>)}
</personInf>
```

return scope of \$i



# Conclusion

- A new technique for pruning nested XQuery queries with composition
- A rule based algorithm for a large XQuery fragment
  - FLWR expression, quantifier
  - Sequence, comparison operation, element construction
  - If-the-else expressions
  - ...
- Experiments show significant gains in evaluation time
- Useful in many contexts (data integration, security)
- Possible extensions:
  - backward axis
  - taking into account document schema



Thank you for your attention