DDO-Free XQuery

Hiroyuki Kato[1], Yasunori Ishihara[2] and Torsten Grust[3]

[1]National Institute of Informatics
[2]Osaka University
[3]Universität Tübingen

The 16th International Symposium on
Database Programming Languages (*DBPL*2017)

September 1, 2017, Munich, Germany

DDO (Distinct Document Order)

"heavily-ordered"

- A prominent feature in XML processing using XQuery
  - ✓ reflecting the models of ordered trees
  - ✓ *the document order*: a total order defined over all nodes in the tree, determined by a preorder traversal

$e/\alpha{::}\tau$ = distinct-doc-order(**for** \$fs:dot **in** e **return** $\alpha{::}\tau$)

- It is potentially costly

$e/\alpha_1{::}\tau_1/\ldots/\alpha_K{::}\tau_K$

= ddo(**for** \$fs:dot **in** ddo(…) **return** $\alpha_K{::}\tau_K$)

- There are studies on avoiding the need for DDO operations.

[SIGMOD02a], [SIGMOD02b], [SIGMOD03], [DBPL03], [DEXA05], [MSCS15], …

DDO-free XQuery

$e/\alpha::\tau$ **=** **for** \$fs:dot **in** e **return** $\alpha::\tau$

Syntactic restriction is needed.

A single-node child-traversal expression \$*v*/**child**::τ is DDO-free

for \$*v* **in** doc("foo.xml")/r/a
return (<u>\$*v*/b</u>, <u>\$*v*/c</u>)

navigating to child nodes from a single node does not require DDO

when XML documents are stored in a serialized (preorder-based) fashion.

Many XQuery engines such as BaseX, MonetDB/XQuery, DB2/pureXML adopt this.

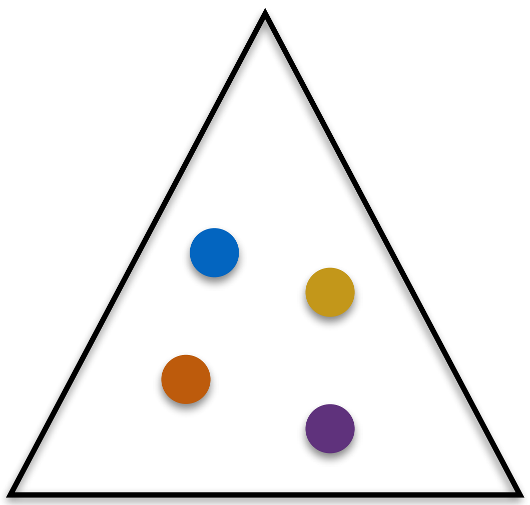
Most theoretical work on XQuery [TODS06], [ICFP15], [ESOP15]
are based on this syntactic restriction.

Twig Query with DDO

Twig queries: one of the typical use cases of XQuery

Extracting subtrees that satisfy tree patterns

Twig queries with DDO are the norm and not the exception.



[[*e*]]=(● ● ● ● ● ● ●)

[[ddo(e)]] = (● ● ● ●)

e may not be DDO-free

[An example] Analyzing a log file in XML format using a twig query

Extracting nodes that satisfying a tree pattern and are sorted in DDO

[A running example]

ddo(**for** *$b* **in** doc("foo.xml")/a/b **return**

for *$a* **in** *$b*/.. **return** (*$b*, *$a*)/c)

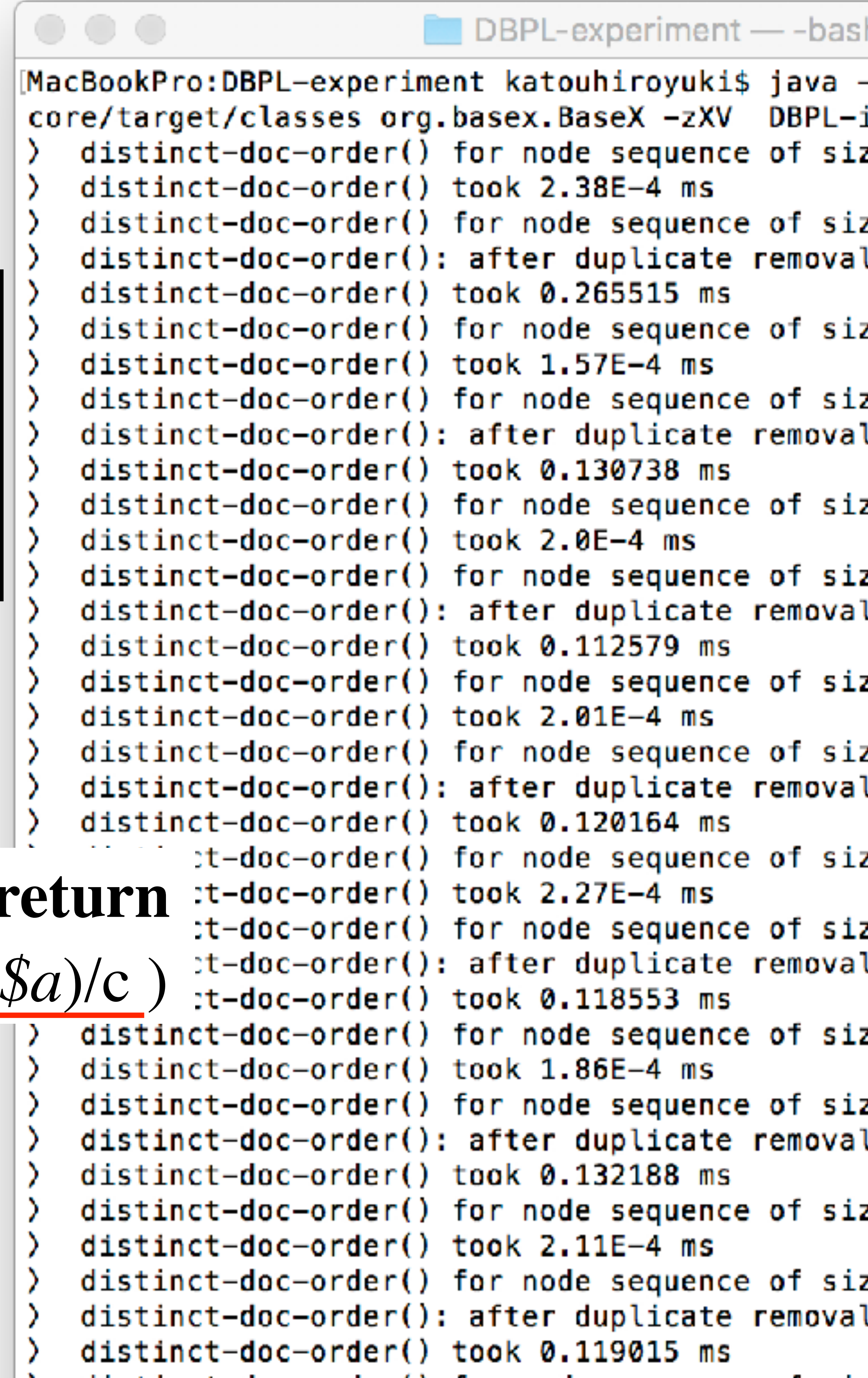
The Problem

Naive evaluation of ddo(e) requires multiple application of ddo
=> may lead to significant sorting overhead

[A running example]

ddo(**for** *$b* **in** doc("foo.xml")/a/b **return**
  for *$a* **in** *$b*/.. **return** (*$b*, *$a*)/c)

DDO-free *e'*



What we have done

An XQuery transformation to evaluate twig queries with DDO efficiently.

[Input]

- a schema information (nested-relational DTD) and
- a twig query *e*

[Output]

a transformed XQuery *e'* such that *e'*=ddo(*e*) and *e'* is DDO-free.

The input XQuery form does not include element constructors because we focus on twig queries, which extract subtrees that satisfy given tree patterns described in XQuery

Basic Idea

$\boxed{\textit{Generate-and-Test}\text{ approach}}$

(G): Prepare a DDO-free skeleton query *s*, which has the ability to *generate all nodes in DDO* for any XML document that conforms to the schema.

(T): Formulate a *s*[*cond*] by injecting node test conditions *cond* extracted from the input query *e*.

$\boxed{s[cond]\text{ is DDO-free}}$

(1) How to prepare the skeleton query *s*.

(2) How to extract appropriate conditions from the input query.

(3) How to inject those conditions into the skeleton query.

Our Solution

(1) How to prepare a DDO-free skeleton query *s*.

- ✓ can be derived for a given nested-relational DTD.

(2) How to extract appropriate conditions from the input query

- ✓ the input query *e* can be transformed into *e'* which has a structure *similar* to that of the skeleton query *s* to reveal the conditions.
- ✓ this transformation *preserves equivalence up to DDO*, ddo(*e*)=ddo(*e'*)

(3) How to inject those conditions into the skeleton query

- ✓ Since *e'* has a structure *similar* to that of *s*, *s*[*cond*] can be obtained in a systematic way.

$\boxed{s[cond] \text{ is DDO-free and equivalent to the input query } e \text{ up to DDO}}$

cond is extracted from *e'*, which is equivalent to *e* up to DDO

Nested-relational DTD

Definition 5 (Nested-relational DTD (NRDTD)). A DTD $D = (\Sigma, l_0, \mu)$ is an NRDTD if D is non-recursive, and $\mu(l)$ is a sequence $(r_1, \ldots, r_N)$ such that each r_i has the form l_i, l_i^*, l_i^+, or $l_i?$, and all l_is are distinct labels.

The height of trees are fixed

Node order can be distinguished by using the single-node child-traversal expression

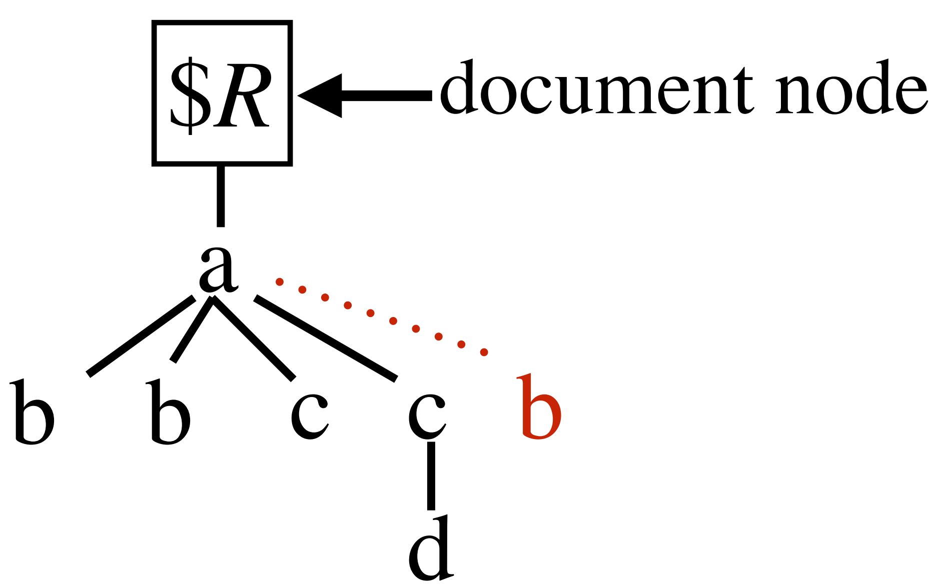
an example: D_1=(Σ_1, a, μ_1)

Σ_1={a,b,c,d}

μ_1(a)=(b*, c+)

μ_1(c)=d?

μ_1(b)=μ_1(d)=()



for \$*a* **in** \$*R*/a
return (\$*a*/b, \$*a*/c)

μ_1'(a)=(b*, c+, b)

Skeleton Query Construction

The design policy

- A skeleton query is a DDO-free expression (use single-node child traversal) ,
- the query encodes the schema that the input document adhere, and
- stub if-conditionals are placed in appropriate positions to control whether an element is produced or not.

An empty sequence () represents **false**.
Any non-empty sequence represents **true**.

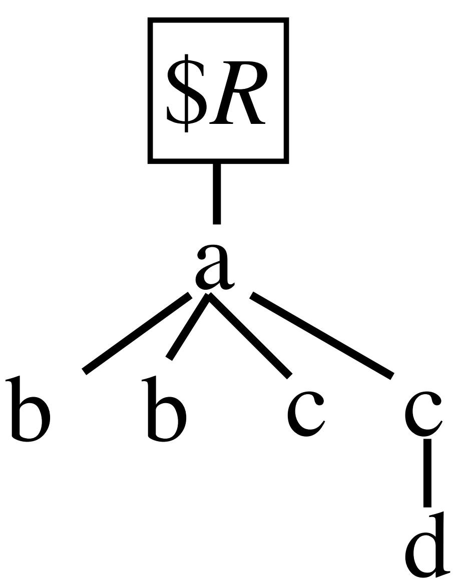
$D_1=(\Sigma_1, \mathrm{a}, \mu_1)$

$\Sigma_1=\{\mathrm{a,b,c,d}\}$

$\mu_1(\mathrm{a})=(\mathrm{b}^*, \mathrm{c}+)$

$\mu_1(\mathrm{c})=\mathrm{d}?$

$\mu_1(\mathrm{b})=\mu_1(\mathrm{d})=()$




```
for $a in $R/a return
   (if () then $a else (),
    for $b in $a/b return (if () then $b else ()),
    for $c in $a/c return
      (if () then $c else (),
       for $d in $c/d return (if () then $d else ())))
```

The skeleton query returns *all nodes in DDO* if the conditions are replaced with **true**.

Our Solution

(1) How to prepare a DDO-free skeleton query *s*.

- ✓ can be derived for a given nested-relational DTD.

(2) How to extract appropriate conditions from the input query

- ✓ the input query *e* can be transformed into *e'* which has **a structure *similar* to that of the skeleton query *s*** to reveal the conditions.
- ✓ this transformation *preserve equivalence up to DDO*, ddo(*e*)=ddo(*e'*)

(3) How to inject those conditions into the skeleton query

- ✓ Since *e'* has a structure *similar* to that of *s*, *s*[*cond*] can be obtained in a systematic way.

s[*cond*] is DDO-free and equivalent to the input query *e* up to DDO

cond is extracted from *e'*, which is equivalent to *e* up to DDO

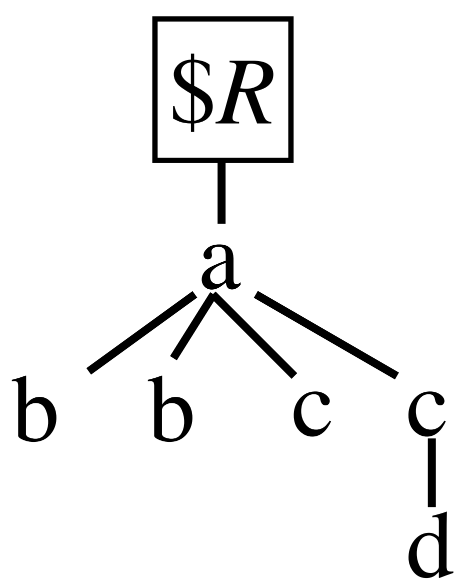
Structural Features of Skeleton Query

(s1) If a node is output it has been previously bound to an output variable,
(s2) all occurrences of **for** are consecutive-child-axis **for**-expressions, and
(s3) a (stub) if-conditional is located in the **return** part of each **for**.

if () **then** $*v* **else** ()

An output variable is bound to nodes to be output.

A consecutive-child-axis **for**-expression is a nested for-expression in which the **in** part is a step expression ($*v*/**child**::τ) and $*v* is defined in the innermost outer **for**.



for $*a* **in** $*R*/a **return**
(**if** () **then** $*a* **else** (),
for $*b* **in** $*a*/b **return** (**if** () **then** $*b* **else** ()),
for $*c* **in** $*a*/c **return**
(**if** () **then** $*c* **else** (),
for $*d* **in** $*c*/d **return** (**if** () **then** $*d* **else** ())))

The Target Form of the Transformed exp.

A sequence expression (e_1, …, e_K), in which e_i is a **for**-expression or $\$R$

It has the following structural features when e_i is a **for**-expression;

(t1) If a node is output it has been previously bound to an output variable,

(t2) all occurrences of **for** are consecutive-child-axis **for**-expressions, and

(t3) if-conditionals that appear in the innermost **return** part of a **for**.

if *cond* **then** $\$v$ **else** ()

Structural features of the skeleton query

(s1) If a node is output it has been previously bound to an output variable,

(s2) all occurrences of **for** are consecutive-child-axis **for**-expressions, and

(s3) a (stub) if-conditional is located in the **return** part of each **for**.

if () **then** $\$v$ **else** ()

Input Query Transformation

The input query *e*

for *$b* **in** $*R*/a/b **return**
    for *$a* **in** *$b*/.. **return** (*$b*, *$a*)/c

up to DDO

The transformed query *e'*

(**for** $*v* **in** $*R*/a **return**
    for $*b* **in** $*v*/b **return**
        for $*o* **in** $*b*/c **return**
            if (**if** $*R*/a **then** $*o* **else** ()) **then** $*o* **else** (),
for $*v* **in** $*R*/a **return**
    for $*o* **in** $*v*/c **return**
        if (**if** (**if** $*R*/a **then** $*R*/a/b **else** ()) **then** $*o* **else** ())
        then $*o*
        else ())

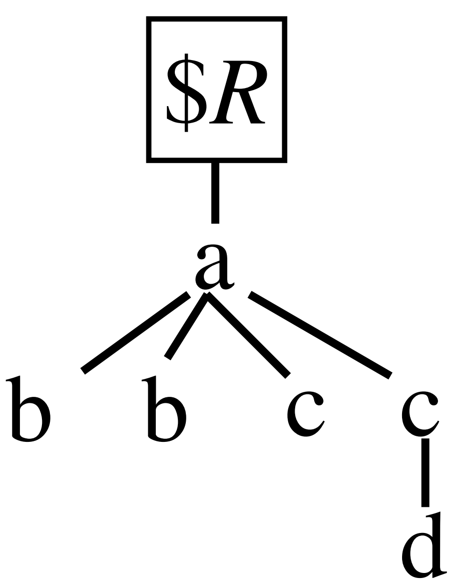
The transformed query e'

```
(for $v in $R/a return
   for $b in $v/b return
      for $o in $b/c return
         if (if $R/a then $o else ()) then $o else (),
 for $v in $R/a return
   for $o in $v/c return
      if (if (if $R/a then $R/a/b else ()) then $o else ())
      then $o
      else ())
```

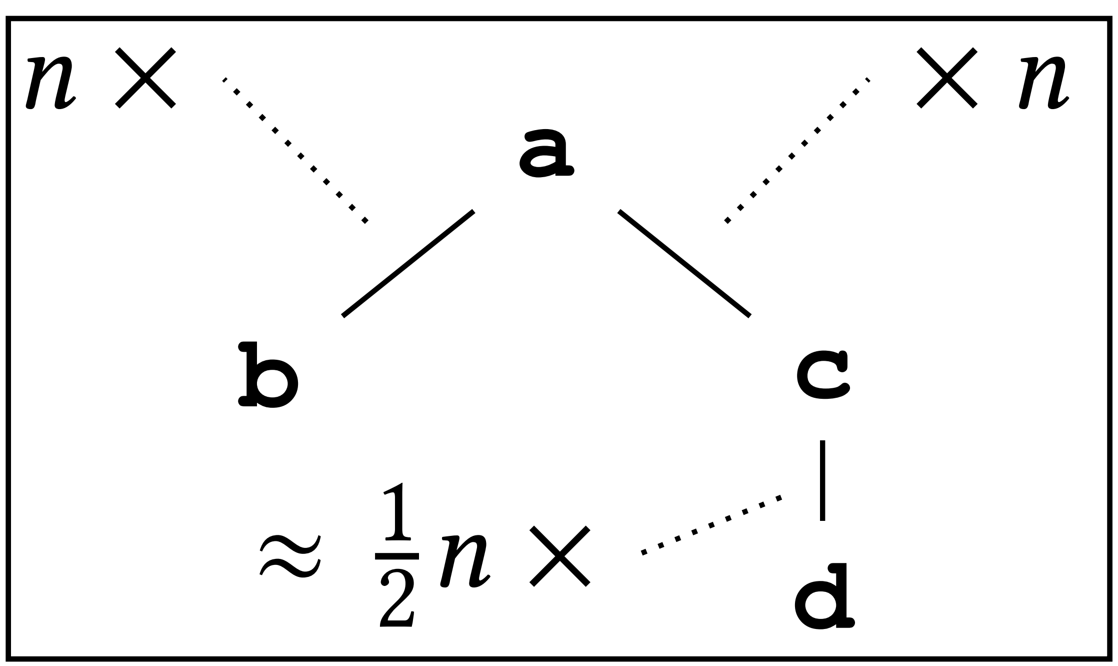
cond

The skeleton query s

```
for $a in $R/a return
   (if () then $a else (),
    for $b in $a/b return (if () then $b else ()),
    for $c in $a/c return
      (if () then $c else (),
       for $d in $c/d return (if () then $d else ())))
```



	BaseX 8.4		Saxon-HE 9.7.0.18J	
doc. size n	original	DDO-free	original	DDO-free
1	1.78	1.06	0.56	1.02
10	7.03	2.17	2.69	3.12
100	40.43	5.30	10.70	6.05
1 000	454.20	17.44	287.67	13.53
10 000	OOM	30.69	79646.54	62.15
100 000	OOM	72.80	OOM	217.07
1 000 000	OOM	404.79	OOM	1531.95



Wall-clock times (msec)
OOM: out of memory
Intel Core i7 3.3GHz, 16GB RAM
Both engines are implemented in Java
Heap size of JVM: 4GB

Conclusion and Future Work

For a given NRDTD and an XQuery *e* for an XML document that conforms to the DTD, *e* can be transformed into *e'* such that *e'*=ddo(*e*) and *e'* is DDO-free.

- **Any XQuery engines can utilize this results.**
- May have some benefits in unordered mode.

Future work:

- A real world practical performance measurement.
- Relaxing the restriction on schemas and input syntax.

Our input syntax

e	::=	$\$v \mid (e, e, \ldots, e) \mid () \mid e/\alpha{::}\tau \mid$ **for** $\$v$ **in** e **return** e
	|	**if** e **then** e **else** $()$
α	::=	**child** | **parent** | **self** | **descendant** | **ancestor**
	|	**descendant-or-self (dos)** | **ancestor-or-self (aos)**
τ	::=	label | $*$