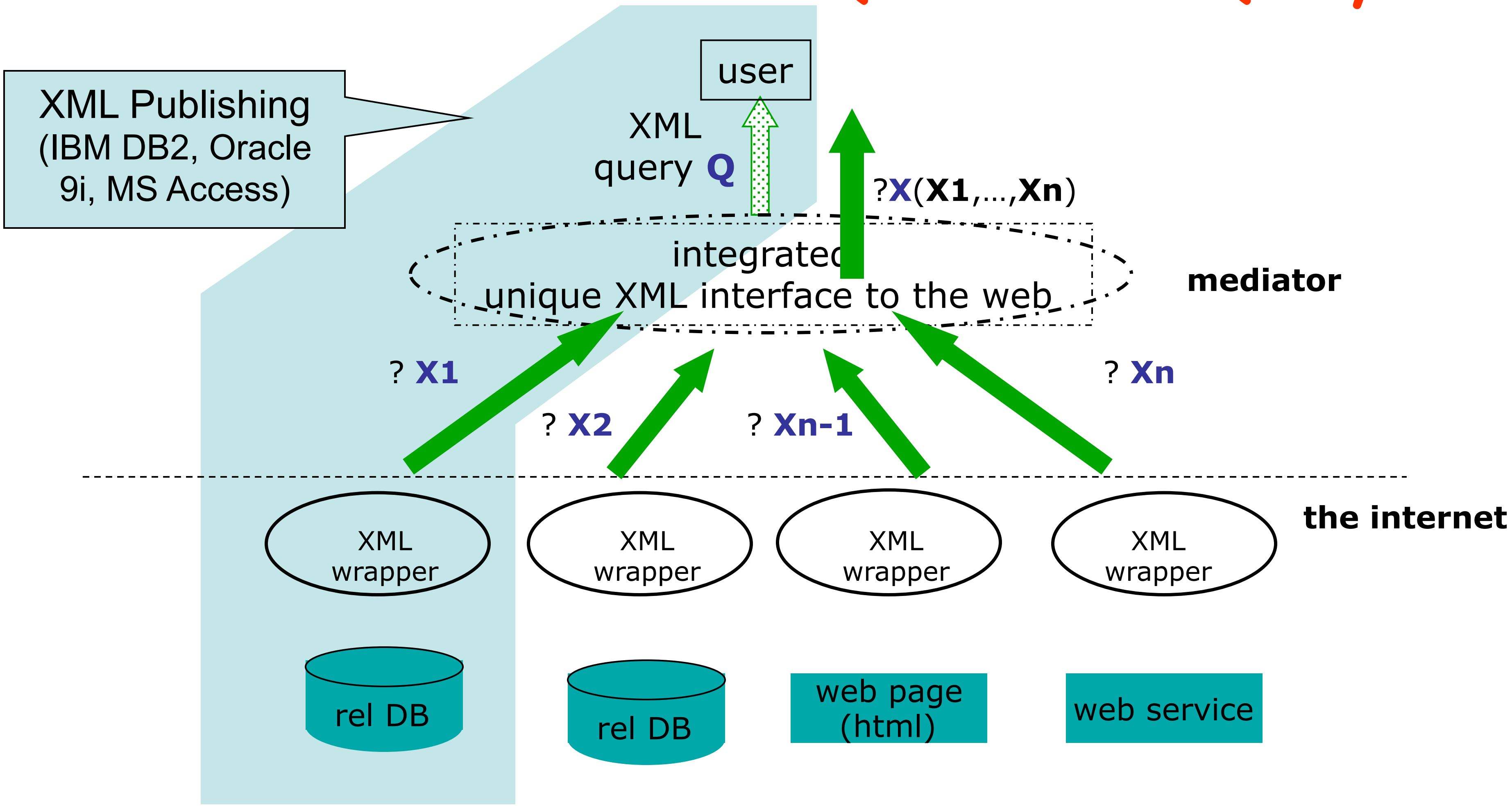
XQuery Advanced Topics

Alin Deutsch

Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization

The Web as Database Queried in XQuery



XML Publishing
(IBM DB2, Oracle
9i, MS Access)

user

XML
query **Q**

?**X**(**X1**,…,**Xn**)

integrated
unique XML interface to the web

mediator

? **X1**

? **X2**

? **Xn-1**

? **Xn**

the internet

XML
wrapper

XML
wrapper

XML
wrapper

XML
wrapper

rel DB

rel DB

web page
(html)

web service

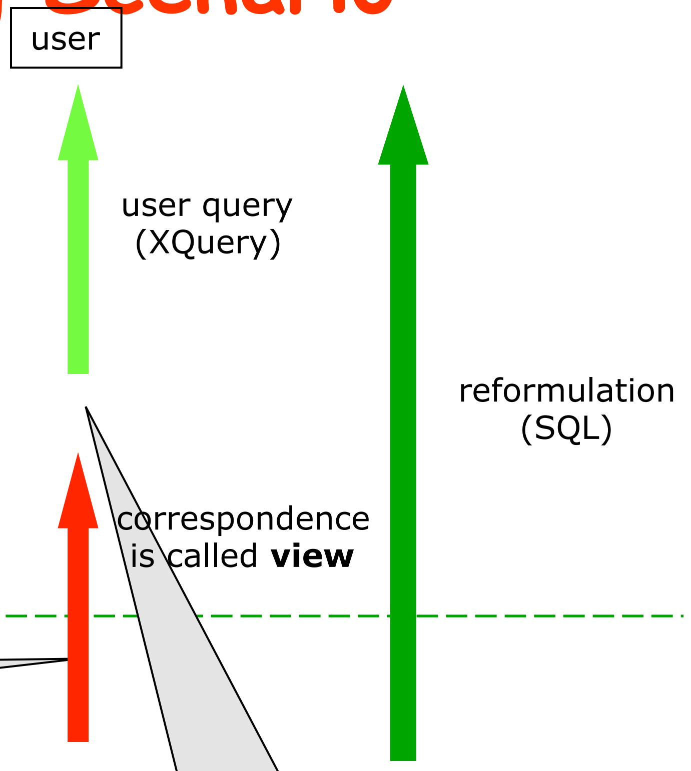
Q, X, X1, …, Xn are XQueries

A Simple Publishing Scenario

virtual data

patient name is hidden

```
<study>
   <case>
     <diag>migraine</diag>
     <drug>aspirin</drug>
     <usage>2/day</usage>
   </case>
   <case>
     <diag>allergy</diag>
     <drug>cortisone</drug>
     <usage>3/day</usage>
    </case>
</study>
```



published data

proprietary data

How to express the view?

prescription

usage	drug	name
2/day	aspirin	John
3/day	cortisone	Jane

patient

name	diagnosis
John	migraine
Jane	allergy

How to “compose” the user query with the view, obtaining the reformulation?

Encoding relational data as XML

Want to specify view from proprietary → published data as XML → XML view expressed in XQuery

prescription

usage	drug	name
2/day	aspirin	John
3/day	cortisone	Jane

patient

name	diagnosis
John	migraine
Jane	allergy

```
<prescription>
    <tuple><usage>2/day</usage>
             <drug>aspirin</drug>
             <name>John</name>
    </tuple>
    <tuple><usage>3/day</usage>
             <drug>cortisone</drug>
             <name>Jane</name>
     </tuple>
</prescription>
```

```
<patient>
    <tuple><name>John</name>
             <diag>migraine</diag>
    </tuple>
    <tuple><name>Jane</name>
             <diag>allergy</diag>
     </tuple>
</patient>
```

Proprietary→Published View: XML → XML

public.xml

```
<study>
    <case><diag>migraine</diag><drug>aspirin</drug>
             <usage>2/day</usage>
    </case>
    <case><diag>allergy</diag><drug>cortisone</drug>
             <usage>3/day</usage>
     </case>
</study>
```

published data

proprietary data

view
expressible
as XQuery

```
<prescription>
    <tuple><usage>2/day</usage>
             <drug>aspirin</drug><name>John</name>
    </tuple>
    <tuple><usage>3/day</usage>
             <drug>cortisone</drug><name>Jane</name>
     </tuple>
</prescription>
```

encoding.xml

The View

```
<study>
  for       $t1 in document(“encoding.xml”)//patient/tuple,
                $n1 in $t1/name/text(),
                $di  in $t1/diagnosis/text(),

            $t2 in document(“encoding.xml”)//prescription/tuple,
                $n2 in $t2/name/text(),
                $dr  in $t2/drug/text(),
                $u   in $t2/usage/text(),
  where
            $n1=$n2
  return
            <case><diag>$di</diag>
                  <drug>$dr</drug>
                  <usage>$u</usage>
            <case>
</study>
```

A Client Query

Find high-maintenance illnesses (require drug usage thrice a day):

```
<results>
  for      $c in document(“public.xml”)//case,
               $d in $c/diag/text(),
               $u in $c/usage/text(),
  where
           $u=“3/day”
  return
           <drug>$d</drug>
</results>
```

Not directly executable, public.xml does not exist

The Reformulated Query

Directly executable, expressed in SQL against the proprietary database:

```
Select     pr.drug

From       patient pa, prescription pr

Where      pa.name = pr.name and
            pr.usage = “3/day”
```

prescription

usage	drug	name
2/day	aspirin	John
3/day	cortisone	Jane

patient

name	diagnosis
John	migraine
Jane	allergy

Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization

XQuery Semantics: Navigation & Tagging

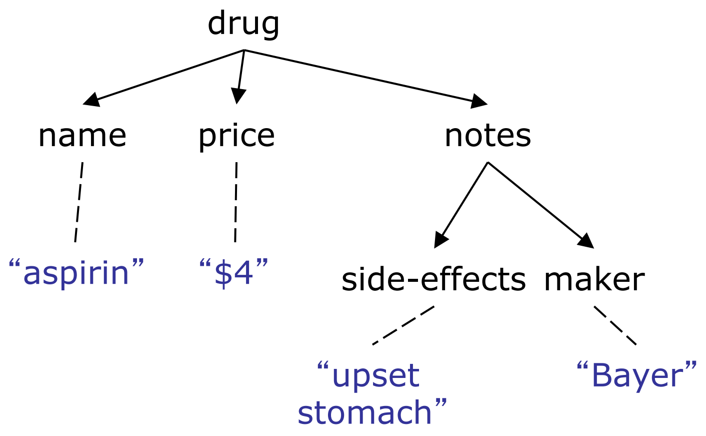
XML data model is a tagged **tree**

```
<drug>
    <name>aspirin</name>
    <price>$4</price>
    <notes>
      <side-effects>upset stomach</side-effects>
      <maker>Bayer</maker>
    </notes>
</drug>
```

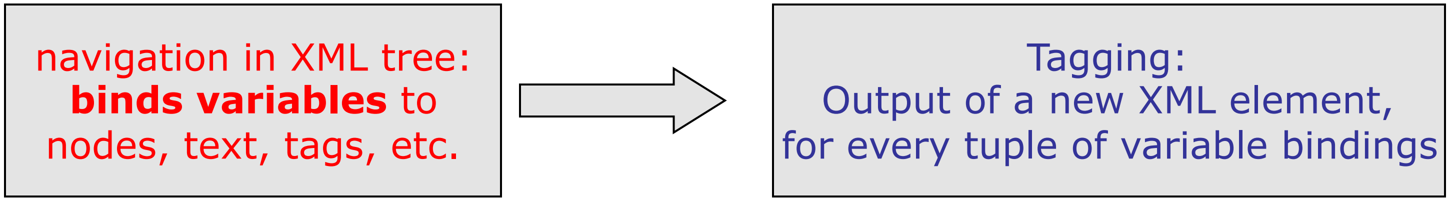
opening tag

text

matching closing tag



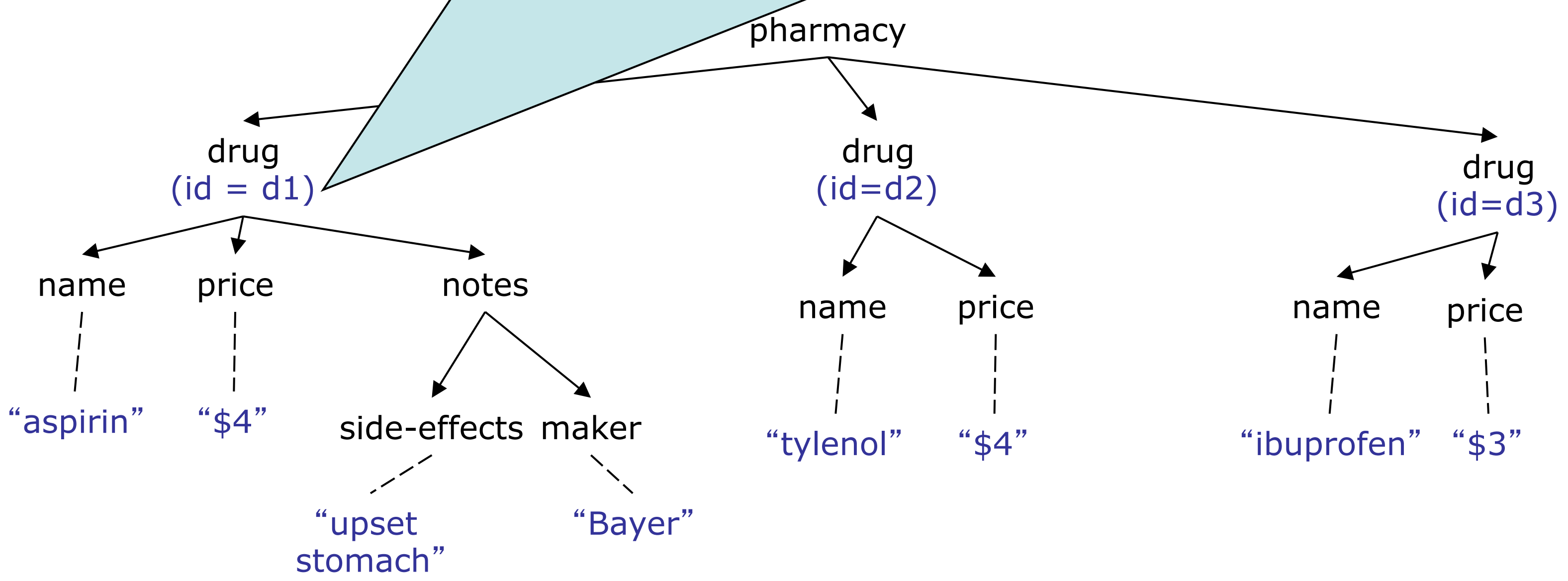
XQueries compute in two stages:



navigation in XML tree: **binds variables** to nodes, text, tags, etc.

Tagging: Output of a new XML element, for every tuple of variable bindings

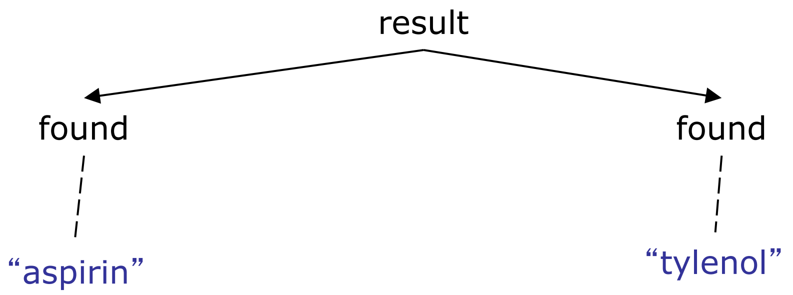
Node identity, for example java reference of DOM node.
Do not confuse with ID attribute.



```
let $d = document(“drugs.xml”)
<result>
  for     $x in $d//drug, $n in $x//name/text(),
          $p in $x//price/text()
  where $p = “$4”
  return
           <found>$n</found>
</result>
```

$x	$n	$p
d1	“aspirin”	“$4”
d2	“tylenol”	“$4”
~~d3~~	~~“ibu”~~	~~“$3”~~

XQuery Semantics: Tagging



```
let $d = document(“drugs.xml”)
<result>
  for      $x in $d//drug, $n in $x//name/text(),
           $p in $x//price/text()
  where $p = “$4”
  return
            <found>$n</found>

</result>
```

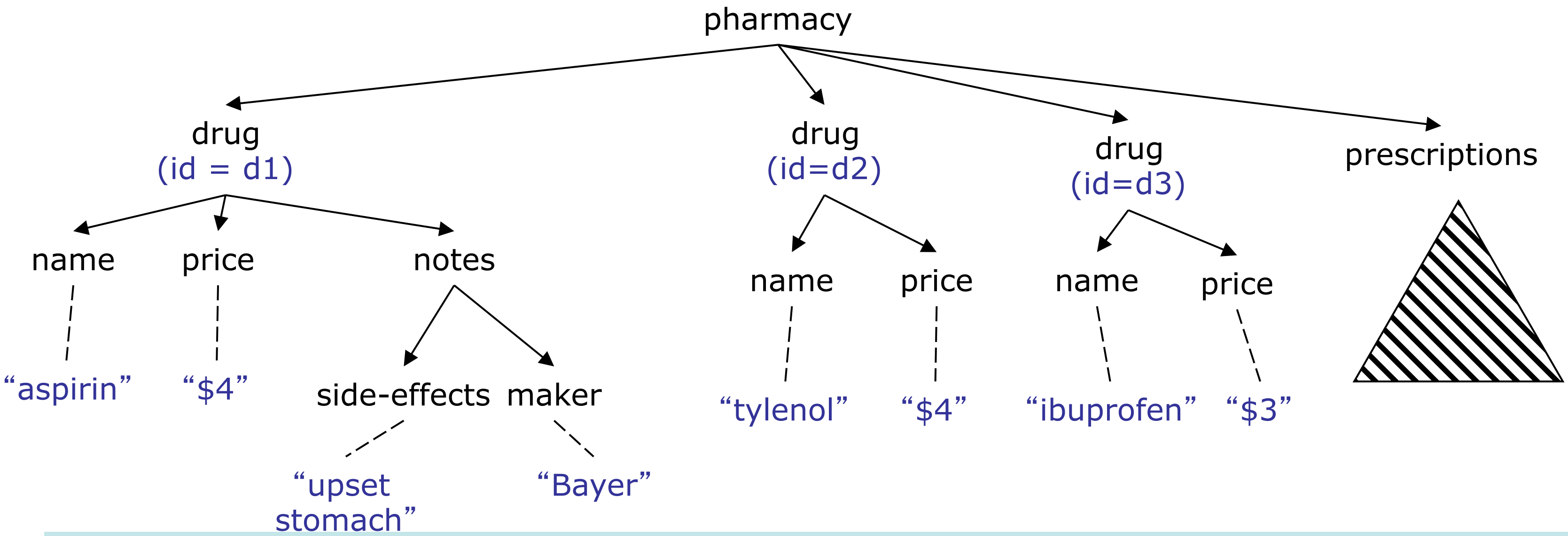
$x	$n	$p
d1	“aspirin”	“$4”
d2	“tylenol”	“$4”

Descendant Navigation

Direct implementation of descendant navigation is wasteful:

<u>for</u> $x <u>in</u> $d//drug

Go to all descendants of the root (all elements), keep <drug>-tagged ones

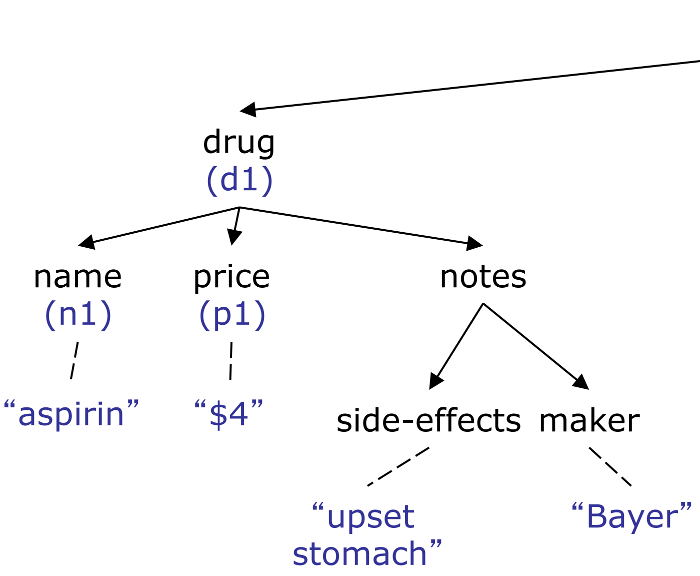
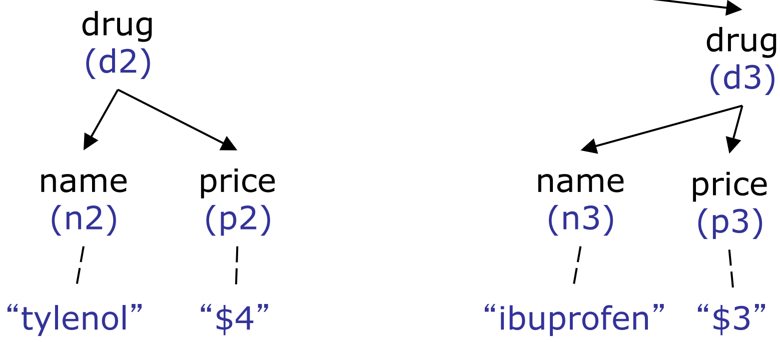


To find the 3 <drug> elements, a direct implementation visits **all elements** in the document (e.g. <notes>). The full query does so repeatedly.
In general, a query with n descendant steps may visit |doc size|^n elements!

Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
  - Index-based
  - Stream-based
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization

Index-based Evaluation

Idea 1: keep an index (associative array, hash table) associating tags with lists of node ids. Allows random access into XML tree.

idx:

tag	node ids
drug	d1,d2,d3
name	n1,n2,n3
price	p1,p2,p3

lookup operation: idx[price] = [p1,p2,p3]

Index-based Evaluation (2)

idx:

tag	node ids
drug	d1,d2,d3
name	n1,n2,n3
price	p1,p2,p3

lookup operation: idx[price] = [p1,p2,p3]

```
foreach $p in idx[price]                          // p1, p2, p3
   if $p/text() = “$4”                            // p1, p2
      foreach $x in idx[drug]                     // d1, d2, d3
         if $p descendant_of $x                   // p1 of d1, p2 of d2
            foreach $n in idx[name]               // n1, n2, n3
               if $n descendant_of $x             // n1 of d1, n2 of d2
               return <found>$n</found>
```

Only 9 elements visited, regardless of size of irrelevant XML subtrees.

But doesn’t the implementation of descendant_of require more visiting?

Ancestor-Descendant Testing in O(1)

Idea 2: identify each node n by a pair of integers pre(n),post(n), with

pre(n) = the rank of n in the preorder traversal of the tree
post(n) = the rank of n in the postorder traversal

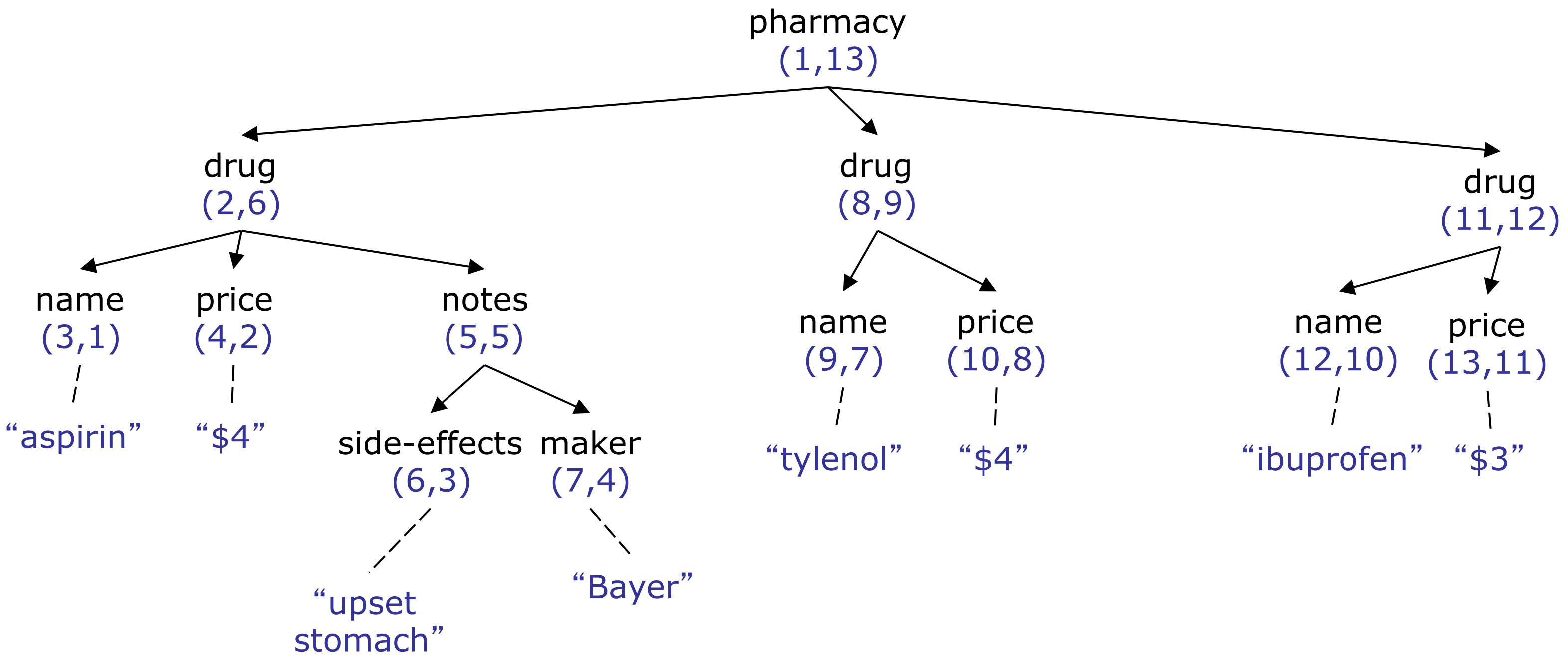
Then

d is descendant of a

↔

pre(d) >= pre(a) and post(d) <= post(a)

Example post-preorder node ids



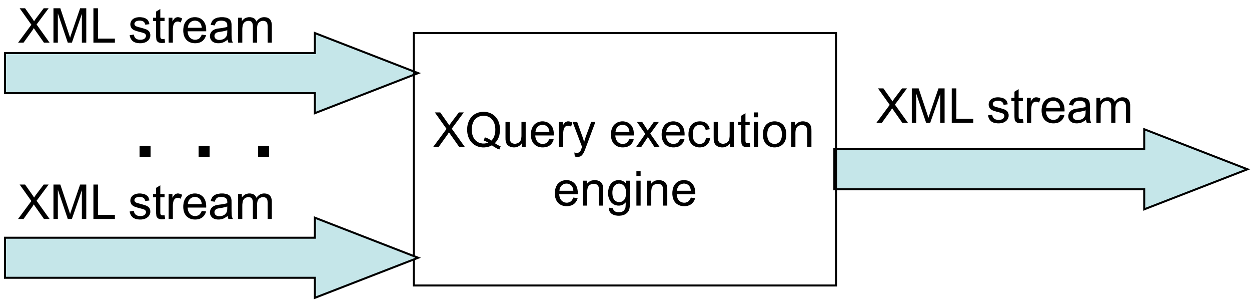
Additional advantage: node identity independent of particular in-memory representation of DOM objects.

Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
  - Index-based
  - Stream-based
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization

Stream-based XQuery Execution

- So far, we assumed construction of DOM tree in memory.
- XML documents can be XML representations of databases. The DOM approach does not scale to typical database sizes.
- We want an execution model that minimizes the memory footprint of the XQuery engine.



Applications of Stream-based Execution

- Besides scaling to database sizes. There are applications where the data is inherently received in streamed form:
- Sensor networks (attend faculty candidate Sam Madden's talk)
- Network monitoring/XML packet routing
- XML document publish/subscribe systems

Stream-based XML Parsing

- A parser generates a stream of predefined events (according to the standard SAX API)
- Applications consume these events.
- Each event triggers a handler. The application is coded by providing the code for the handlers.

```
XML input to parser                stream of events output by parser

<a>                                open("a")
 <b>                               open("b")
   <c>                             open("c")
     someText                      text("someText")
   </c>                            close("c")
 </b>                              close("b")
 <d>                               open("d")
 moreText                          text("moreText")
 </d>                              close("d")
</a>                               close("a")
```

- A free SAX parser: http://xml.apache.org/xerces-j/

Stream-Based XQuery Navigation

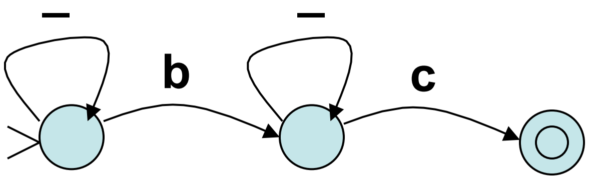
Idea: turn path expressions into Finite Automata over alphabet containing the set of element tags

E.g.

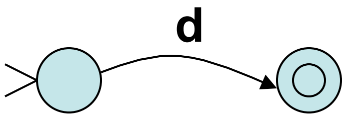
```
for $x in //b//c, $y in $x/d
```

compiles to

$x:



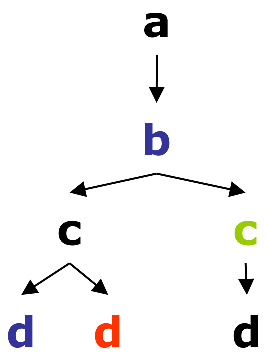
$y:



Only one automaton active at any moment.

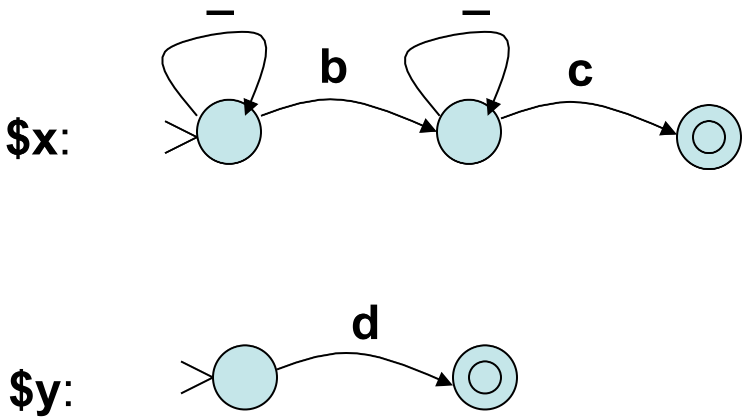
Automaton of $y is active only as long as that of $x is in final state

Matching XPaths Against Streams

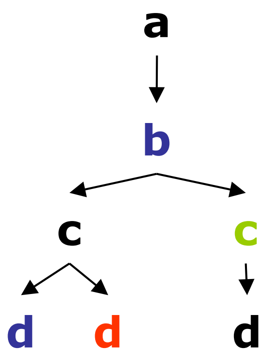


o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)

for $x in //b//c, $y in $x/d

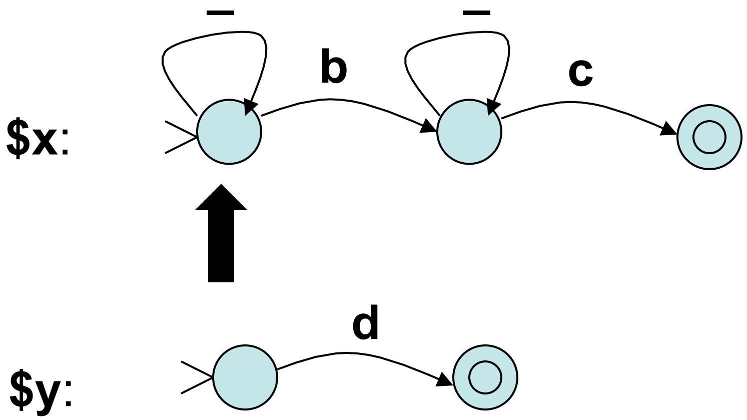


Matching XPaths Against Streams

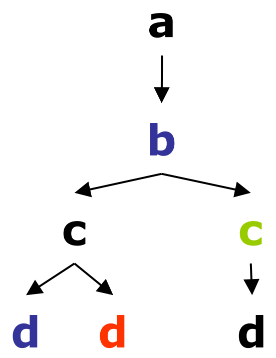


o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)

for $x in //b//c, $y in $x/d

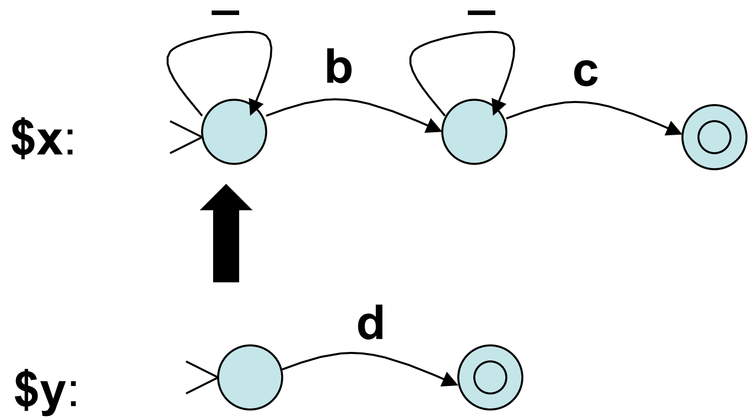


Matching XPaths Against Streams

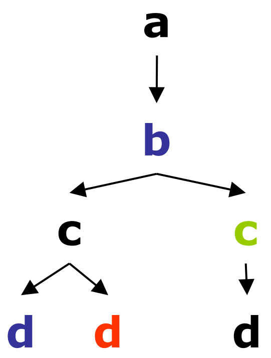


o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)

for $x in //b//c, $y in $x/d

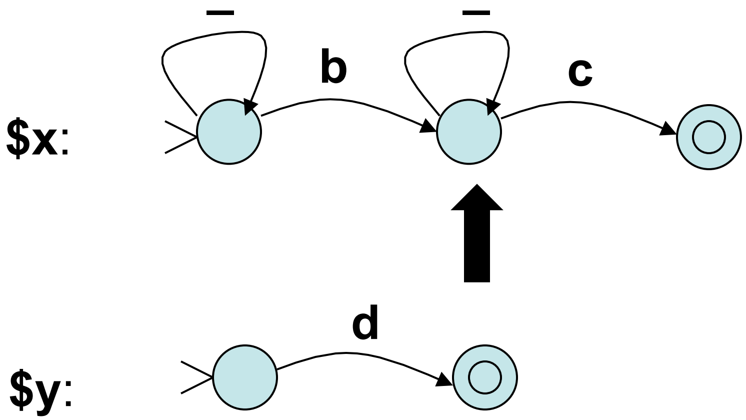


Matching XPaths Against Streams

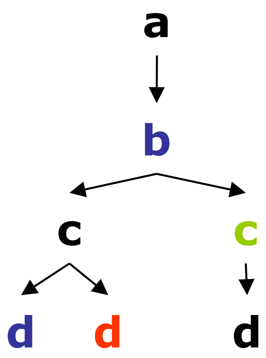


o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)

for $x in //b//c, $y in $x/d

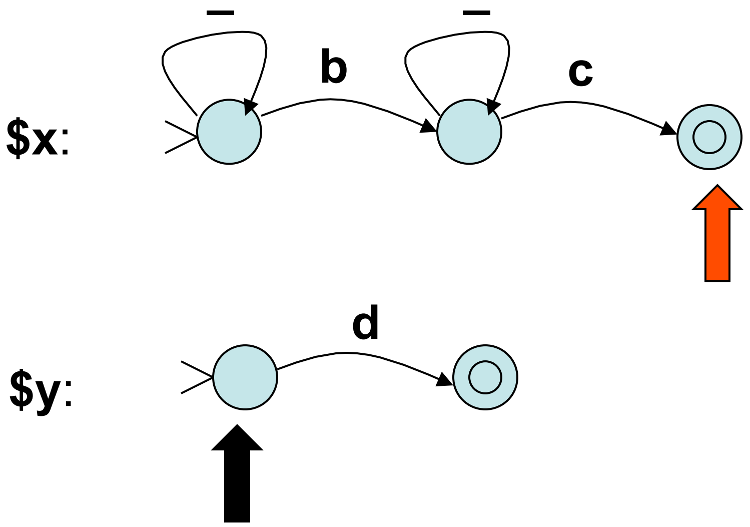


Matching XPaths Against Streams

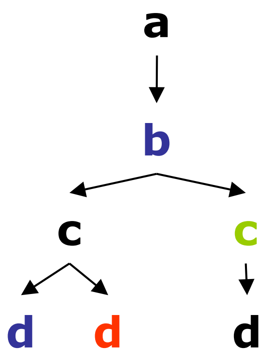


o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)

for $x in //b//c, $y in $x/d

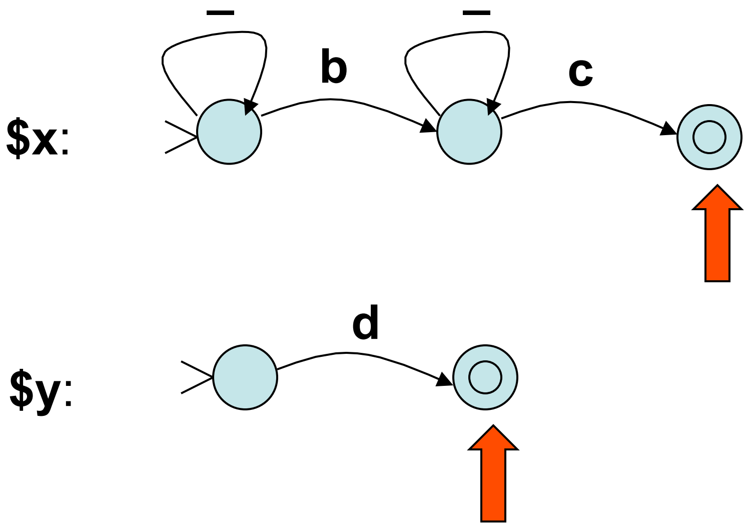


Matching XPaths Against Streams



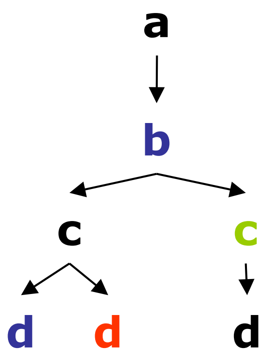
o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)

for $x in //b//c, $y in $x/d



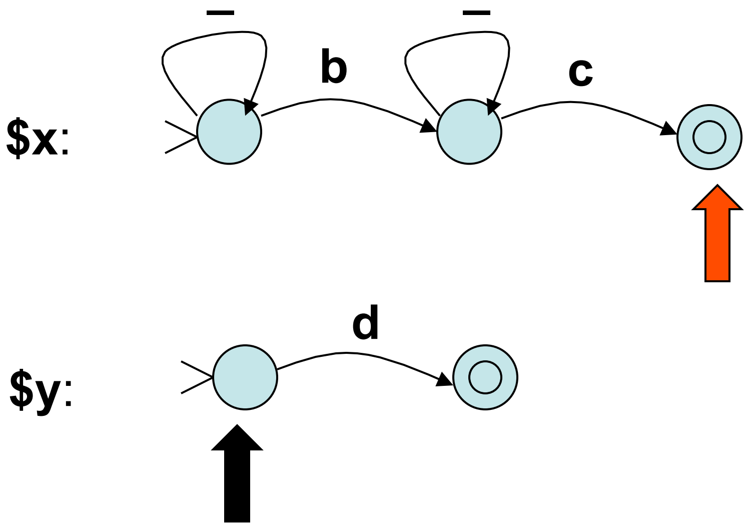
Need to reset automaton for $y

Matching XPaths Against Streams

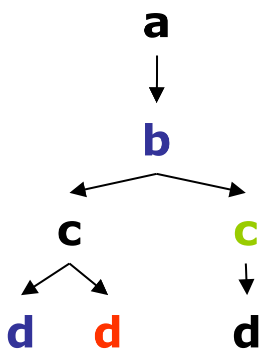


o(a),
o(b),
o(c), o(d), c(d), o(d), c(d), c(c),
o(c), o(d), c(d), c(c),
c(b),
c(a)

for $x in //b//c, $y in $x/d



Matching XPaths Against Streams



o(a),

o(b),

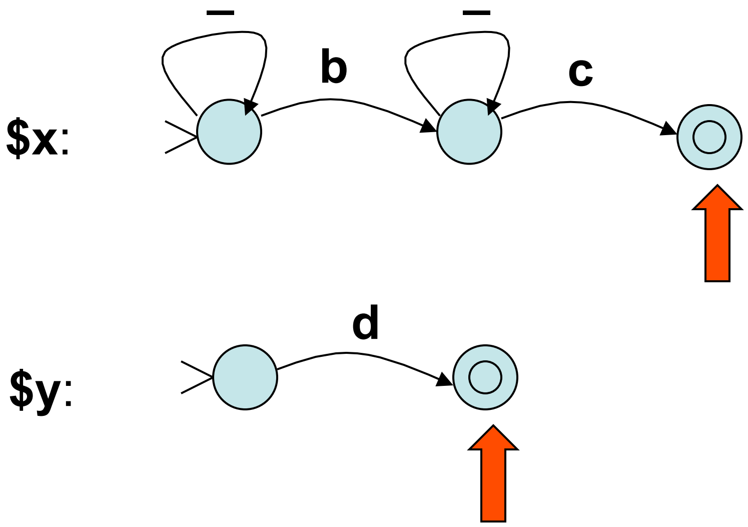
o(c), o(d), c(d), o(d), c(d), c(c),

o(c), o(d), c(d), c(c),

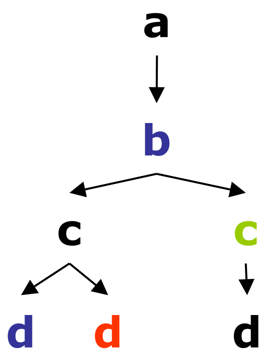
c(b),

c(a)

for $x in //b//c, $y in $x/d

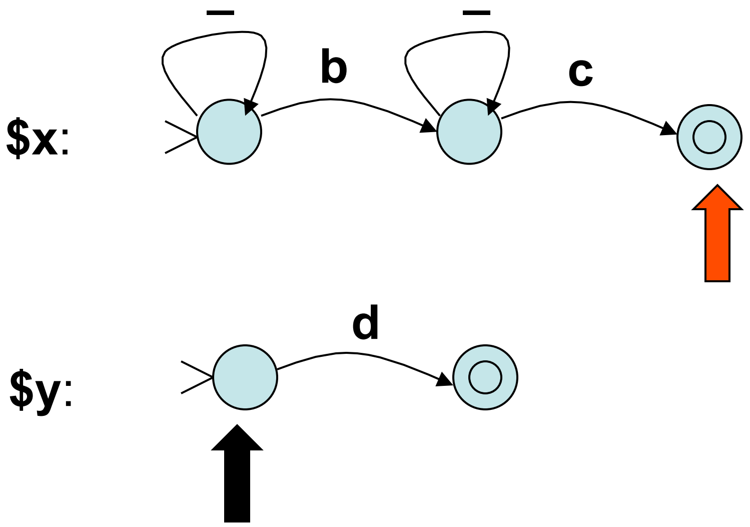


Matching XPaths Against Streams



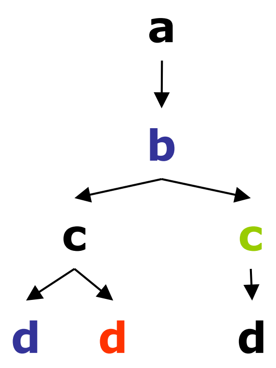
o(a),
  o(b),
    o(c), o(d), c(d), o(d), c(d), c(c),
    o(c), o(d), c(d), c(c),
  c(b),
c(a)

for $x in //b//c, $y in $x/d



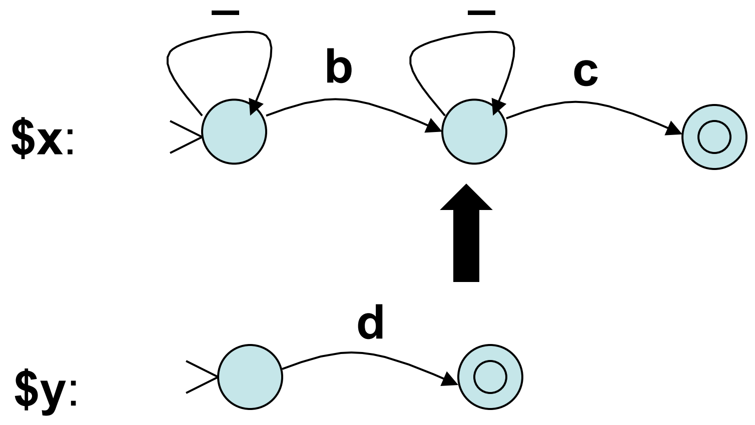
Need to reset automaton for $x to state prior to reading black c element

Matching XPaths Against Streams



```
o(a),
  o(b),
    o(c), o(d), c(d), o(d), c(d), c(c),
    o(c), o(d), c(d), c(c),
  c(b),
c(a)
```

`for $x in //b//c, $y in $x/d`



Automaton Extended with Stack

Let d be the transition function of automaton A. The corresponding extension of A with a stack is defined as follows:

current state	current event in stream	stack action	next state
Q	open(tag)	push(Q)	d(Q)
Q	close(tag)	Q’=pop()	Q’

Convince yourselves that the run of this automaton on the stream in the example corresponds to the intended sequence of states.

An additional use of PDAs, aside from parsing.

Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization

Semantic Optimization

- Sometimes, we can translate away descendant computation.
- Consider the following DTD describing the structure of drug.xml

```
<!ELEMENT pharmacy (drug*)>
<!ELEMENT drug (name,price,notes?)>
```

- Then for all documents satisfying DTD:

`for $x in $d//drug, $n in $x//name/text()` is equivalent to
`for $x in $d/drug, $n in $x/name/text()`

Semantic Optimization As Typechecking

For all XML documents conforming to the DTD

```
<!ELEMENT pharmacy (drug*)>
<!ELEMENT drug (name,price,notes?)>
```

we can determine statically that

```
for $x in $d//drug, $m in $d/maker
```

returns the empty answer.

Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization

Element Equality in XQuery

- Two kinds of equality:
  - "==" id-based (an element node is equal only to itself)
  - "=" value-based

- Value-based equality underwent several drafts,

- Initially (about one year into standardization process):
  text-centric point of view. XML elements are value-equal iff their text values are equal after stripping away the XML annotations.

  E.g. <a><b>f</b><c>oo</c></a> = <m>foo</m>

- Currently:
  XML elements are equal iff their corresponding trees are isomorphic

Id-based Element Equality

Let $x be bound to an XML tree. Then

<a>$x</a>

creates a new XML tree (fresh node ids) and it is short for

<a>recursive copy of $x</a>

Always true:

(<a>$x</a>)/a/* = $x (value-based equality)

Always false:

(<a>$x</a>)/a/* == $x (id-based equality)

Roadmap

- Use of XQuery for Web Data Integration
- XQuery Evaluation Models
- Optimization
- Flavor of Standardization Issues
  - Equality in XQuery
- More on Optimization

More on XQuery Optimization

- There are many ways to write the same query (i.e. there are many distinct XQuery expressions with identical semantics)

- Some of these expressions lead to cheaper execution than their counterparts.

- Goal of query optimization:
  given a query Q, find the optimal query Q' with identical semantics
  (we say that Q and Q' are equivalent)

- Basic test in query optimization: checking query equivalence

- The more expressive a language, the harder it is to test equivalence

- Various classes of XQueries have distinct complexity:

  PTIME (1), NP-complete (1), Π_2^p-complete (4), PSPACE-complete (1), EXPTIME-complete, undecidable

The UCSD Database Lab

- Main Focus: XML Query Optimization
- Check out the weekly DB Research Meeting
- Faculty
  - Victor Vianu
  - Yannis Papakonstantinou
  - Alin Deutsch

- San Diego SuperComputer Resaerchers
  - Ilkay Altintas
  - Amarnath Gupta

www.db.ucsd.edu