Schemas for Safe and Efficient XML Processing

Dario Colazzo
Université Paris Sud - INRIA

Giorgio Ghelli
Università di Pisa

Carlo Sartiani
Università della Basilicata

Tutorial Outline

- Overview of schema languages
  - DTD
  - XML Schema
  - Relax NG
- Applications of schemas
  - Type inference
  - Projection
  - Query/update independence

Tutorial Outline

- Advanced topics
  - Strong determinism
  - CHAREs
  - Conflict-freedom
- Open problems and concluding remarks

OVERVIEW OF SCHEMA LANGUAGES

Motivation

- XML documents should be typed to be safely and effciently processed
- Advantages
  - Query and update optimization
  - Type-checking of applications
  - Needed for data integration and data exchange

Schema Languages

- Many schema languages: DTDs, XML Schema, Relax-NG, etc
- Based on a common formal model
  - Regular expressions describing the content model of an element
  - Regular expression types

DTDs

- DTDs (Document Type Definitions) [BP+06] is the first schema formalism for XML data
- A DTD associates a regular expression to **each tag**
- The regular expression describes the content model of each element with that tag
- We can use standard regular expressions

$$r ::= \epsilon \quad | \quad a \quad | \quad r + r \quad | \quad r \cdot r \quad | \quad r^* \quad | \quad r^+ \quad | \quad r?$$

DTD Example

- The famous bib DTD:

```
<!ELEMENT bib  (book* )>

<!ELEMENT book  (title,  (author+ | editor+ ), publisher, price )>

<!ATTLIST book  year CDATA  #REQUIRED >

<!ELEMENT author  (last, first )>

<!ELEMENT editor  (last, first, affiliation )>

<!ELEMENT title  (#PCDATA )>
```

Restrictions

- According to the W3C regular expressions must be 1-unambiguous
- Informally, while matching a word against a regular expression, no backtrack or lookahead is required
- Formally [BKW98]
  - A regular expression r on the alphabet $\sum$ is 1-unambiguous if and only if there not exist $u,v,w \in \sum^*$ such that $ua_iv \in L(r)$ and $ua_jw \in L(r)$, where a_i and a_j are two distinct occurrences of a inside r
- Example
  - $(a + b)^*.a$ is 1-ambiguous
  - $(a + b)^+.c.a$ is 1-unambiguous

More on 1-unambiguity

- 1-unambiguity is a semantic restriction and has no easy syntactical characterization
  - This may lead to many errors

- A regular expression can be unambiguous but 1-ambiguous
  - ab + ac is not ambiguous (both ab and ac have unique parsing trees)
  - ab + ac is 1-ambiguous (we need a lookahead to correctly parse ab or ac)

Complexity of Decisional Problems

DTD REs	Membership	Inclusion	Intersection
1-ambiguous	PTIME	PSPACE [SM73]	PSPACE [K77]
1-unambiguous	PTIME	PTIME [H10]	PSPACE [MNS04]

XML Schema

- XML Schema [FW04, TB+04, BM04] is a schema language defined by the W3C
- Main objectives
  - To overcome the 1-type restriction of DTDs
  - To use more flexible regular expressions
  - To offer more datatypes
    - string
    - boolean
    - date
    - etc.

XML Schema Principles

- As in a DTD, the content model of an element is described by a regular expression
- We can have multiple element types with the same tag
- Elements with the same tag in the same content model must have the same type
- The type of an element is identified by the path from the root

XML Schema Example

- A snippet of the XMark XSD

```
<xs:element name="site">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="regions"/>
            <xs:element ref="categories"/>
            <xs:element ref="catgraph"/>
            <xs:element ref="people"/>
            <xs:element ref="open_auctions"/>
            <xs:element ref="closed_auctions"/>
         </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:element name="categories">
      <xs:complexType>
         <xs:sequence>
            <xs:element ref="category" maxOccurs="unbounded"/>
         </xs:sequence>
      </xs:complexType>
</xs:element>
```

XML Schema Regular Expressions

- In XML Schema we can use standard regular expressions extended with
  - Counting: `minOccurs` - `maxOccurs`
  - Unordered concatenation: `all` groups

$$r ::= \epsilon \quad | \quad a \quad | \quad r + r \quad | \quad r \cdot r \quad | \quad r[m..n] \quad | \quad \&(a_1, \ldots, a_n)$$

Counting

- Through counting we can specify the minimal and maximal number of occurrences of an instance of a regular expression
  - **`minOccurs`** specifies the minimal number of occurrences
  - **`maxOccurs`** specifies the maximal number of occurrences

```
<xs:element ref="category" maxOccurs="unbounded"/>
```

- More formally

$$[\![r\,[m..n]]\!] \quad = \quad \cup_{i=m..n} [\![r]\!]^i$$

All Groups

- In XML Schema regular expressions we can also specify a limited form of unordered concatenation on leaf nodes through `all` groups
- Example (from Wikipedia XSD)

```
<complexType name="DiscussionThreadingInfo">
        <all>
            <element name="ThreadSubject" type="string" />
            <element name="ThreadParent" type="positiveInteger" />
            ...
            <element name="ThreadAuthor" type="string" />
        </all>
</complexType>
```

- Formally

$$[\![\&(a_1, \ldots, a_n)]\!] = \bigcup_{\sigma_i \in Perm(\{1,\ldots,n\})} [\![a_{\sigma_1}]\!] \cdot \ldots \cdot [\![a_{\sigma_n}]\!]$$

Restrictions

- Regular expressions must satisfy the infamous UPA (Unique Particle Attribution) constraint

  "A content model must be formed such that during *validation* of an element information item sequence, the particle component contained directly, indirectly or *implicitly* therein with which to attempt to *validate* each item in the sequence in turn can be uniquely determined without examining the content or attributes of that item, and without any information about the items in the remainder of the sequence."

- This is commonly interpreted as 1-unambiguity [SM04].

Complexity of Decisional Problems

XSD REs	Membership	Inclusion	Intersection
1-ambiguous	PTIME [KT03]	EXPSPACE [GMN07]	EXPTIME [GMN07]
1-unambiguous	PTIME	coNP-hard [KT03]	PSPACE-hard [GGM09]

Relax-NG

- Relax-NG [CM01] is a schema language designed by James Clark and Makoto Murata
- Relax-NG objectives:
  - Fast membership algorithms
  - Only minor restrictions
    - 1-unambiguity is not required
  - The use of externally defined datatypes

Relax-NG Content Models

- Relax-NG content models are defined according to the following class of regular expressions

$$r ::= \epsilon \quad | \quad a \quad | \quad r + r \quad | \quad r \cdot r \quad | \quad r \& r \quad | \quad r^* \quad | \quad r^+ \quad | \quad r?$$

- Unlike in XML Schema, & here is the interleaving operator
- & can be used to combine generic regular expressions
- Formally

$$v \& w \quad \stackrel{def}{=} \quad \{v_1 \cdot w_1 \cdot \ldots \cdot v_n \cdot w_n \quad | \quad v_1 \cdot \ldots \cdot v_n = v,\ w_1 \cdot \ldots \cdot w_n = w,\ v_i \in \Sigma^*,\ w_i \in \Sigma^*,\ n > 0\}$$

$$[\![r_1 \& r_2]\!] \quad \stackrel{def}{=} \quad \bigcup_{w_1 \in [\![r_1]\!],\ w_2 \in [\![r_2]\!]} w_1 \& w_2$$

Example

Example

```
<element xmlns="http://relaxng.org/ns/structure/1.0"
name="museum">
 <interleave>
  <group>
   <element name="group1.member1"><empty/></element>
   <element name="group1.member2"><empty/></element>
  </group>
  <group>
   <element name="group2.member1"><empty/></element>
   <element name="group2.member2"><empty/></element>
  </group>
 </interleave>
</element>
```

Example

```
<element xmlns="http://relaxng.org/ns/structure/1.0"
name="museum">
 <interleave>
  <group>
   <element name="group1.member1"><empty/></element>
   <element name="group1.member2"><empty/></element>
  </group>
  <group>
   <element name="group2.member1"><empty/></element>
   <element name="group2.member2"><empty/></element>
  </group>
 </interleave>
</element>
```

```
<museum>
 <group2.member1/>
 <group1.member1/>
 <group2.member2/>
 <group1.member2/>
</museum>
```

Example

```
<element xmlns="http://relaxng.org/ns/structure/1.0"
name="museum">
 <interleave>
  <group>
   <element name="group1.member1"><empty/></element>
   <element name="group1.member2"><empty/></element>
  </group>
  <group>
   <element name="group2.member1"><empty/></element>
   <element name="group2.member2"><empty/></element>
  </group>
 </interleave>
</element>
```

```
<museum>
 <group2.member1/>
 <group1.member1/>
 <group2.member2/>
 <group1.member2/>
</museum>
```

```
<museum>
 <group2.member2/>
 <group1.member1/>
 <group2.member1/>
 <group1.member2/>
</museum>
```

Example

```
<element xmlns="http://relaxng.org/ns/structure/1.0"
name="museum">
 <interleave>
  <group>
   <element name="group1.member1"><empty/></element>
   <element name="group1.member2"><empty/></element>
  </group>
  <group>
   <element name="group2.member1"><empty/></element>
   <element name="group2.member2"><empty/></element>
  </group>
 </interleave>
</element>
```

```
<museum>
 <group2.member1/>
 <group1.member1/>
 <group2.member2/>
 <group1.member2/>
</museum>
```

~~<museum>~~
~~ <group2.member2/>~~
~~ <group1.member1/>~~
~~ <group2.member1/>~~
~~ <group1.member2/>~~
~~</museum>~~

Restrictions

- The only restrictions concern the interleave operator

  "For a pattern **`<interleave> p1 p2 </interleave>`**,

  - there must not be a name that belongs to both the name class of an element pattern referenced by a ref pattern occurring in **`p1`** and the name class of an element pattern referenced by a ref pattern occurring in **`p2`**, and
  - a text pattern must not occur in both **`p1`** and **`p2`**."

- More formally, in r_1 & r_2 $names(r_1) \cap names(r_2) = \varnothing$

Complexity of Decision Problems

Membership	Inclusion	Intersection
PTIME	PSPACE-hard	PSPACE-hard

APPLICATIONS

FROM TYPE INFERENCE TO QUERY AND UPDATE OPTIMIZATION

Querying and updating XML data

Querying and updating XML data

- First W3C recommendation of XML in 1998; then many query language proposals (Lorel, XQL, XML-QL, XPath, Quilt)

Querying and updating XML data

- First W3C recommendation of XML in 1998; then many query language proposals (Lorel, XQL, XML-QL, XPath, Quilt)
- In 2001, first W3C working draft for **XQuery**, encapsulating several features of predecessors
  - functional language
  - navigation expressed by means of XPath
  - powerful for-let-where-return core

Querying and updating XML data

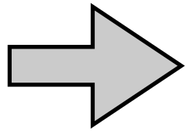
- First W3C recommendation of XML in 1998; then many query language proposals (Lorel, XQL, XML-QL, XPath, Quilt)
- In 2001, first W3C working draft for **XQuery**, encapsulating several features of predecessors
  - functional language
  - navigation expressed by means of XPath
  - powerful for-let-where-return core
- Recently enriched with update mechanisms insert-rename-delete-replace

Querying and updating XML data

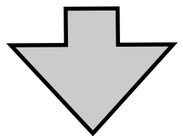
- First W3C recommendation of XML in 1998; then many query language proposals (Lorel, XQL, XML-QL, XPath, Quilt)
- In 2001, first W3C working draft for **XQuery**, encapsulating several features of predecessors
  - functional language
  - navigation expressed by means of XPath
  - powerful for-let-where-return core
- Recently enriched with update mechanisms insert-rename-delete-replace
- *De facto* standard XML query and update language

An XQuery query

```
<bib>
   <book>
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
   </book>
   <book>
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
    <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>
</bib>
```



```
for $x in /descendant::book/child::author
return  <res>
                $x/child::first,
                $x/child::last
          </res>
```



```
<res>
     <first>W.</first> <last>Stevens</last>
</res>
<res>
      <first>W.</first><last>Stevens</last>
</res>
```

Type Inference

- Performed at static time
  - Input: a query Q and input schema Sin
  - Output: a schema S' containing all possible results of Q overs Sin
- Example: the type editor* is inferred for Q=//editor over bib docs.
- Recursive process: type of Q determined in terms of types of subexpressions of Q
- Usually defined by means of deduction rules

$$\frac{\text{premise}_1 \quad \ldots\ldots \quad \text{premise}_n}{\text{conclusion}}$$

What it is Useful For?

- Checking correctness of transformations Q from S_{in} to S_{out}
  - First S' is **inferred** from Q and S_{in}, and then S'<S_{out} is checked
  - Complexity of schema inclusion goes from PSPACE to EXPSPACE, depending on type mechanisms in S' and S_{out}, but PTIME under restrictions.
- Projecting XML documents [BCCN'06, BBCMS'11]
- Detecting XML query-update independence [BC'09, BCU'12]

Main Properties

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}
- Precision:

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}
- Precision:
  - no exact typing in the realm of REs

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}
- Precision:
  - no exact typing in the realm of REs
  - Q=(/a, ,/a) over r→a* generates $\{a^n,b,a^n \mid n \geq 0\}$

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}
- Precision:
  - no exact typing in the realm of REs
  - Q=(/a, ,/a) over r→a* generates $\{a^n,b,a^n \mid n \geq 0\}$
  - Inferred type a*,b,a*

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}
- Precision:
  - no exact typing in the realm of REs
  - Q=(/a, ,/a) over r→a* generates {a^n,b,a^n| n≥0}
  - Inferred type a*,b,a*
  - Precision is crucial for avoiding false negatives: (a+b)* should be avoided

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}
- Precision:
  - no exact typing in the realm of REs
  - Q=(/a, ,/a) over r→a* generates {a^n,b,a^n| n≥0}
  - Inferred type a*,b,a*
  - Precision is crucial for avoiding false negatives: (a+b)* should be avoided
- Complexity: exponential cost for high precision

Main Properties

- Soundness: inferred type includes all possible query results over S_{in}
- Precision:
  - no exact typing in the realm of REs
  - Q=(/a, ,/a) over r→a* generates $\{a^n,b,a^n \mid n \geq 0\}$
  - Inferred type a*,b,a*
  - Precision is crucial for avoiding false negatives: (a+b)* should be avoided
- Complexity: exponential cost for high precision
- Simplicity: avoid involved type rules: soundness needs to be formally proved.

Type Inference

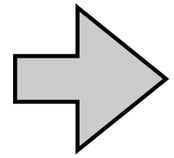
- In a nutshell, type inference ~ mimicking evaluation of Q over S_{in}
- More or less difficult depending on S_{in}.
- Easy task on simple S_{in}, like {a→b,c ; b,c→String}
- What about more complex ones?
  - S_{in}={a→(b | c)* ; b,c→String}
- Need of operators over types, mimicking basic query semantics operation.

Type Inference Mechanisms

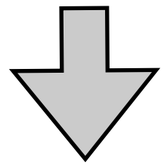
- Main ones for: axis navigation, NT filtering, for-query iteration
- For simplicity: here, focus on DTDs, child and descendant navigation, no element construction.
- A DTD over a finite alphabet $\sum$ is S=(d,s) where:
  - d associates a regular expression over $\sum \cup$ {String} to each element tag in $\sum$
  - s is the root element tag
- Inferred type for Q: regular expression R (some times noted as T) whose symbols are defined by d

Example

bib → (book*)
book → (title, (author+ |editor+), publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String



```
for $x in /bib/book/author
return  $x/first, $x/last
```



(first, last) *

The **result** type describes query results

Typing axis navigation

Typing axis navigation

- Child navigation can be simulated by the function d.

Typing axis navigation

- Child navigation can be simulated by the function d.
- Descendant navigation is trickier:
  - $des_S(a)=(\alpha 1 | \ldots . | \alpha n)^*$
  - $\{\alpha 1, \ldots ., \alpha n\}$ biggest set of type symbols in $\Sigma \cup \{String\}$ descending from a according to d definitions.
- example
  - $des_S(bib)=(book | author | \ldots\ldots | price | String)^*$

Typing axis navigation

- Child navigation can be simulated by the function d.
- Descendant navigation is trickier:
  - $des_S(a)=(\alpha 1 | \ldots . | \alpha n)^*$
  - $\{\alpha 1, \ldots ., \alpha n\}$ biggest set of type symbols in $\sum \cup \{String\}$ descending from a according to d definitions.
  - example
    - $des_S(bib)=(book \mid author \mid \ldots\ldots \mid price \mid String)^*$
- Any idea about a more precise typing for descendant navigation?

Typing NT

Typing NT

- Notation R::NT→R'
- Defined via simple structural induction

Typing NT

- Notation R::NT→R'
- Defined via simple structural induction

(R1,R2)::NT	→	R1::NT, R1::NT		a :: a	→	a
(R1 | R2)::NT	→	R1::NT | R1::NT		a :: b	→	()
R*::NT	→	(R::NT)*		a :: text()	→	()
R+::NT	→	(R::NT)+		α :: node()	→	α
R?::NT	→	(R::NT)?		String::text()	→	String
				String::a	→	a

Typing NT

- Notation R::NT→R'
- Defined via simple structural induction

(R1,R2)::NT	→	R1::NT, R1::NT		a :: a	→	a
(R1 | R2)::NT	→	R1::NT | R1::NT		a :: b	→	()
R*::NT	→	(R::NT)*		a :: text()	→	()
R+::NT	→	(R::NT)+		α :: node()	→	α
R?::NT	→	(R::NT)?		String::text()	→	String
				String::a	→	a

(a | b)*,String :: a → a* (a | b)*,String :: text() → String

(a | b)*,String::c →()

A simple case, let-queries

- Q= let $x:=Q1 return Q2
  - recursive typing: the type of Q determined in terms of those for Q1 and Q2
  - typing of Q2 according to an environment recording the type of $x
  - the rule:

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma, \$x{:}T1 \vdash Q2 : T}{\Gamma \vdash \text{let } \$x{:=}Q1 \text{ return } Q2 : T}$$

- For-queries: can we rely on a similar rule?

Typing for-queries

$$\frac{\Gamma \vdash Q1 : T1 \qquad ???}{\Gamma \vdash \$x \text{ in } Q1 \text{ return } Q2 : T}$$

- Need to simulate iteration over values of T1
- Widely adopted approach: typing via iteration on T1
- Auxiliary judgement: $\Gamma \vdash \$x \text{ in } T \rightsquigarrow Q : T'$
- The rule

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma \vdash \$x \text{ in } T1 \rightsquigarrow Q : T}{\Gamma \vdash \$x \text{ in } Q1 \text{ return } Q2 : T}$$

Typing for-queries

- Main rule

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma \vdash \$x \text{ in } T1 \rightsquigarrow Q : T}{\Gamma \vdash \$x \text{ in } Q1 \text{ return } Q2 : T}$$

- Auxiliary rules

Typing for-queries

- Main rule

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma \vdash \$x \text{ in } T1 \rightsquigarrow Q : T}{\Gamma \vdash \$x \text{ in } Q1 \text{ return } Q2 : T}$$

- Auxiliary rules

$$\frac{\Gamma, \$x{:}a \vdash Q : T}{\Gamma \vdash \$x \text{ in } a \rightsquigarrow Q : T} \qquad \frac{\Gamma, \$x{:}\text{String} \vdash Q : T}{\Gamma \vdash \$x \text{ in String} \rightsquigarrow Q : T}$$

Typing for-queries

- Main rule

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma \vdash \$x \text{ in } T1 \rightsquigarrow Q : T}{\Gamma \vdash \$x \text{ in } Q1 \text{ return } Q2 : T}$$

- Auxiliary rules

$$\frac{\Gamma, \$x{:}a \vdash Q : T}{\Gamma \vdash \$x \text{ in } a \rightsquigarrow Q : T} \qquad \frac{\Gamma, \$x{:}\text{String} \vdash Q : T}{\Gamma \vdash \$x \text{ in String} \rightsquigarrow Q : T}$$

$$\frac{}{\Gamma \vdash \$x \text{ in } () \rightsquigarrow Q : ()} \qquad \frac{\Gamma \vdash \$x \text{ in } R \rightsquigarrow Q : T \quad op \in \{*, +, ?\}}{\Gamma \vdash \$x \text{ in } Rop \rightsquigarrow Q : Top}$$

Typing for-queries

- Main rule

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma \vdash \$x \text{ in } T1 \rightsquigarrow Q : T}{\Gamma \vdash \$x \text{ in } Q1 \text{ return } Q2 : T}$$

- Auxiliary rules

$$\frac{\Gamma, \$x{:}a \vdash Q : T}{\Gamma \vdash \$x \text{ in } a \rightsquigarrow Q : T} \qquad \frac{\Gamma, \$x{:}\text{String} \vdash Q : T}{\Gamma \vdash \$x \text{ in String} \rightsquigarrow Q : T}$$

$$\frac{}{\Gamma \vdash \$x \text{ in } () \rightsquigarrow Q : ()} \qquad \frac{\Gamma \vdash \$x \text{ in } R \rightsquigarrow Q : T \quad op \in \{*, +, ?\}}{\Gamma \vdash \$x \text{ in } Rop \rightsquigarrow Q : Top}$$

$$\frac{\Gamma \vdash \$x \text{ in } R1 \rightsquigarrow Q : T1 \qquad \Gamma \vdash \$x \text{ in } R2 \rightsquigarrow Q : T2}{\Gamma \vdash \$x \text{ in } R1 \mid R2 \rightsquigarrow Q : T1 \mid T2} \qquad \frac{\Gamma \vdash \$x \text{ in } R1 \rightsquigarrow Q : T1 \qquad \Gamma \vdash \$x \text{ in } R2 \rightsquigarrow Q : T2}{\Gamma \vdash \$x \text{ in } R1, R2 \rightsquigarrow Q : T1, T2}$$

Remaining cases

Remaining cases

$$\frac{\$x{:}T \in \Gamma}{\Gamma \vdash \$x : T} \qquad \frac{s \text{ is a string value}}{\Gamma \vdash s : \text{String}} \qquad \frac{}{\Gamma \vdash () : ()} \qquad \frac{\Gamma \vdash Q1 : T1 \quad \Gamma \vdash Q1 : T1\ T2}{\Gamma \vdash Q1, Q2 : T1, T2}$$

Remaining cases

$$\frac{\$x{:}T \in \Gamma}{\Gamma \vdash \$x : T} \qquad \frac{s \text{ is a string value}}{\Gamma \vdash s : \text{String}} \qquad \frac{}{\Gamma \vdash () : ()} \qquad \frac{\Gamma \vdash Q1 : T1 \quad \Gamma \vdash Q1 : T1\ T2}{\Gamma \vdash Q1, Q2 : T1, T2}$$

$$\frac{S=(d,s) \qquad s :: NT \rightarrow T}{\Gamma \vdash /child::NT : T} \qquad \frac{S=(d,s) \quad des_S(s) = R \quad R :: NT \rightarrow T}{\Gamma \vdash /descendant::NT : T}$$

Remaining cases

$$\frac{\$x{:}T \in \Gamma}{\Gamma \vdash \$x : T} \qquad \frac{s \text{ is a string value}}{\Gamma \vdash s : \text{String}} \qquad \frac{}{\Gamma \vdash () : ()} \qquad \frac{\Gamma \vdash Q1 : T1 \quad \Gamma \vdash Q1 : T1\ T2}{\Gamma \vdash Q1, Q2 : T1, T2}$$

$$\frac{S=(d,s) \qquad s :: NT \rightarrow T}{\Gamma \vdash /child::NT : T} \qquad \frac{S=(d,s) \quad des_S(s) = R \quad R :: NT \rightarrow T}{\Gamma \vdash /descendant::NT : T}$$

$$\frac{\$x{:}a \in \Gamma \qquad d(a) = R \quad R :: NT \rightarrow T}{\Gamma \vdash \$x/child::NT : T} \qquad \frac{\$x{:}a \in \Gamma \qquad des_S(a) = R \quad R :: NT \rightarrow T}{\Gamma \vdash \$x/descendant::NT : T}$$

$$\frac{\$x{:}String \in \Gamma}{\Gamma \vdash \$x/axis::NT : ()}$$

Typing in action

Typing in action

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

$$\frac{() \vdash \text{/descendant :: author} :}{() \vdash \text{for \$x in /descendant :: author return \$x/first, \$x/last} :}$$

Typing in action

$$\frac{}{() \vdash \text{/descendant :: author} :}$$

$$\frac{() \vdash \text{/descendant :: author} :}{() \vdash \text{for \$x in /descendant :: author return \$x/first, \$x/last} :}$$

Typing in action

S=(d,bib) des_S(bib)=(book |...| author |....)*

() ⊢ /descendant :: author :

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author→

() ⊢ /descendant :: author :

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ /descendant :: author :

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ /descendant :: author : author*

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ /descendant :: author : author*

() ⊢ $x in author* ⇝ $x/first, $x/last :

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

$S=(d,bib)$ $des_S(bib)=(book\ |...|\ author\ |....)^*$

$(book\ |...|\ author\ |....)^* :: author \rightarrow author^*$

$() \vdash /descendant :: author : author^*$

$() \vdash \$x \text{ in } author \rightsquigarrow \$x/first, \$x/last :$

$() \vdash \$x \text{ in } author^* \rightsquigarrow \$x/first, \$x/last :$

$() \vdash \text{for } \$x \text{ in } /descendant :: author \text{ return } \$x/first, \$x/last :$

Typing in action

$$\frac{S=(d,bib)\quad des_S(bib)=(book\ |...|\ author\ |....)^*\qquad (book\ |...|\ author\ |....)^*::author\rightarrow author^*}{() \vdash /descendant::author : author^*}$$

$$\frac{\$x: author \vdash \ \$x/first, \$x/last :}{() \vdash \$x\ in\ author \rightsquigarrow \$x/first, \$x/last :}$$

$$\frac{() \vdash /descendant::author : author^* \qquad () \vdash \$x\ in\ author^* \rightsquigarrow \$x/first, \$x/last :}{() \vdash for\ \$x\ in\ /descendant::author\ return\ \ \$x/first, \$x/last :}$$

Typing in action

$$\frac{\$x: \text{author} \vdash \$x/\text{first} : \qquad \$x: \text{author} \vdash \$x/\text{last} :}{\$x: \text{author} \vdash \$x/\text{first}, \$x/\text{last} :}$$

$$\frac{S=(d,bib)\ \ des_S(bib)=(\text{book} \mid ... \mid \text{author} \mid ....)^* \qquad (\text{book} \mid ... \mid \text{author} \mid ....)^* :: \text{author} \rightarrow \text{author}^*}{() \vdash /\text{descendant} :: \text{author} : \text{author}^*} \qquad \frac{\$x: \text{author} \vdash \$x/\text{first}, \$x/\text{last} :}{() \vdash \$x \text{ in } \text{author} \rightsquigarrow \$x/\text{first}, \$x/\text{last} :}$$

$$\frac{() \vdash /\text{descendant} :: \text{author} : \text{author}^* \qquad () \vdash \$x \text{ in } \text{author}^* \rightsquigarrow \$x/\text{first}, \$x/\text{last} :}{() \vdash \text{for } \$x \text{ in } /\text{descendant} :: \text{author return } \$x/\text{first}, \$x/\text{last} :}$$

Typing in action

d(author)=last,first

last,first :: first →

$x: author ⊢ $x/first : $x: author ⊢ $x/last :

$x: author ⊢ $x/first, $x/last :

() ⊢ $x in author ⇝ $x/first, $x/last :

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ /descendant :: author : author*

() ⊢ $x in author* ⇝ $x/first, $x/last :

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

d(author)=last,first

last,first :: first → first

$x: author ⊢ $x/first : $x: author ⊢ $x/last :

$x: author ⊢ $x/first, $x/last :

() ⊢ $x in author ⇝ $x/first, $x/last :

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ /descendant :: author : author*

() ⊢ $x in author* ⇝ $x/first, $x/last :

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

d(author)=last,first

last,first :: first → first

$x: author ⊢ $x/first : first $x: author ⊢ $x/last :

$x: author ⊢ $x/first, $x/last :

$x: author ⊢ $x/first, $x/last :

() ⊢ $x in author ⇝ $x/first, $x/last :

() ⊢ $x in author* ⇝ $x/first, $x/last :

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ /descendant :: author : author*

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

$$\dfrac{\dfrac{d(author)=last,first \quad last,first :: first \rightarrow first}{\$x: author \vdash \$x/first : first} \quad \dfrac{d(author)=last,first \quad last,first :: last \rightarrow}{\$x: author \vdash \$x/last :}}{\$x: author \vdash \$x/first, \$x/last :}$$

$$\dfrac{\dfrac{S=(d,bib) \quad des_S(bib)=(book\ |...|\ author\ |....)^* \quad (book\ |...|\ author\ |....)^* :: author \rightarrow author^*}{() \vdash /descendant :: author : author^*} \quad \dfrac{\$x: author \vdash \$x/first, \$x/last :}{() \vdash \$x \text{ in } author \rightsquigarrow \$x/first, \$x/last :}}{() \vdash \$x \text{ in } author^* \rightsquigarrow \$x/first, \$x/last :}$$

$$() \vdash \text{for } \$x \text{ in } /descendant :: author \text{ return } \$x/first, \$x/last :$$

Typing in action

$$\dfrac{\dfrac{d(author)=last,first \quad last,first :: first \rightarrow first}{\$x: author \vdash \$x/first : first} \quad \dfrac{d(author)=last,first \quad last,first :: last \rightarrow last}{\$x: author \vdash \$x/last :}}{\dfrac{\$x: author \vdash \$x/first, \$x/last :}{() \vdash \$x \text{ in } author \rightsquigarrow \$x/first, \$x/last :}}$$

$$\dfrac{S=(d,bib) \quad des_S(bib)=(book\ |...|\ author\ |....)^* \quad (book\ |...|\ author\ |....)^* :: author \rightarrow author^*}{() \vdash /descendant :: author : author^*}$$

$$\dfrac{() \vdash /descendant :: author : author^* \quad () \vdash \$x \text{ in } author^* \rightsquigarrow \$x/first, \$x/last :}{() \vdash \text{for } \$x \text{ in } /descendant :: author \text{ return } \$x/first, \$x/last :}$$

Typing in action

$$\dfrac{\dfrac{\dfrac{\dfrac{}{d(\text{author})=\text{last,first}\quad \text{last,first} :: \text{first} \rightarrow \text{first}}}{\$x: \text{author} \vdash \$x/\text{first} : \text{first}} \quad \dfrac{\dfrac{}{d(\text{author})=\text{last,first}\quad \text{last,first} :: \text{last} \rightarrow \text{last}}}{\$x: \text{author} \vdash \$x/\text{last} : \text{last}}}{\$x: \text{author} \vdash \$x/\text{first}, \$x/\text{last} :} \quad \dfrac{\dfrac{}{S=(d,bib)\quad des_S(bib)=(\text{book} \mid ... \mid \text{author} \mid ....)^*\quad (\text{book} \mid ... \mid \text{author} \mid ....)^* :: \text{author} \rightarrow \text{author}^*}}{() \vdash /\text{descendant} :: \text{author} : \text{author}^*}}{() \vdash \$x \text{ in } \text{author} \rightsquigarrow \$x/\text{first}, \$x/\text{last} :} \quad \dfrac{\dfrac{\dfrac{}{S=(d,bib)\quad des_S(bib)=(\text{book} \mid ... \mid \text{author} \mid ....)^*\quad (\text{book} \mid ... \mid \text{author} \mid ....)^* :: \text{author} \rightarrow \text{author}^*}}{() \vdash /\text{descendant} :: \text{author} : \text{author}^*}}{() \vdash \$x \text{ in } \text{author}^* \rightsquigarrow \$x/\text{first}, \$x/\text{last} :}}{() \vdash \text{for } \$x \text{ in } /\text{descendant} :: \text{author return } \$x/\text{first}, \$x/\text{last} :}$$

Typing in action

$$\dfrac{\dfrac{d(author)=last,first \quad last,first :: first \rightarrow first}{\$x: author \vdash \$x/first : first} \quad \dfrac{d(author)=last,first \quad last,first :: last \rightarrow last}{\$x: author \vdash \$x/last : last}}{\$x: author \vdash \$x/first, \$x/last : first,last}$$

$$\dfrac{\dfrac{S=(d,bib) \quad des_S(bib)=(book\ |...|\ author\ |....)^* \quad (book\ |...|\ author\ |....)^* :: author \rightarrow author^*}{() \vdash /descendant :: author : author^*} \quad \dfrac{\$x: author \vdash \$x/first, \$x/last : first,last}{() \vdash \$x \text{ in } author \rightsquigarrow \$x/first, \$x/last :}}{() \vdash \$x \text{ in } author^* \rightsquigarrow \$x/first, \$x/last :}$$

$$() \vdash \text{for } \$x \text{ in } /descendant :: author \text{ return } \$x/first, \$x/last :$$

Typing in action

d(author)=last,first
last,first :: first → first

d(author)=last,first
last,first :: last → last

$x: author ⊢ $x/first : first

$x: author ⊢ $x/last : last

$x: author ⊢ $x/first, $x/last : first,last

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

$x: author ⊢ $x/first, $x/last : first,last

() ⊢ /descendant :: author : author*

() ⊢ $x in author → $x/first, $x/last : first,last

() ⊢ $x in author* → $x/first, $x/last :

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

d(author)=last,first
last,first :: first → first

d(author)=last,first
last,first :: last → last

$x: author ⊢ $x/first : first

$x: author ⊢ $x/last : last

$x: author ⊢ $x/first, $x/last : first,last

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ $x in author ⇝ $x/first, $x/last : first,last

() ⊢ /descendant :: author : author*

() ⊢ $x in author* ⇝ $x/first, $x/last : (first,last)*

() ⊢ for $x in /descendant :: author return $x/first, $x/last :

Typing in action

d(author)=last,first
last,first :: first → first

d(author)=last,first
last,first :: last → last

$x: author ⊢ $x/first : first

$x: author ⊢ $x/last : last

$x: author ⊢ $x/first, $x/last : first,last

S=(d,bib) des_S(bib)=(book |...| author |....)*

(book |...| author |....)*:: author → author*

() ⊢ /descendant :: author : author*

$x: author ⊢ $x/first, $x/last : first,last

() ⊢ $x in author ⇝ $x/first, $x/last : first,last

() ⊢ $x in author* ⇝ $x/first, $x/last : (first,last)*

() ⊢ for $x in /descendant :: author return $x/first, $x/last : (first,last)*

Precision and complexity [CS'11]

- Case analysis for for-queries ensures high precision
- Precision further augmented for non recursive types with different typing for descendant axis.
- Precision does not come for free: exponential type inference for nested for-queries
- Fortunately, polynomial time in a wide class of practical cases.

The core of several existing type systems

- *A Semi-monad for Semi-structured Data*. Mary F. Fernández, Jérôme Siméon, Philip Wadler, ICDT 2001

- *Types for path correctness of XML queries*. Dario Colazzo, Giorgio Ghelli, Paolo Manghi, Carlo Sartiani, ICFP 2004:

- *Regular Expression Subtyping for XML Query and Update Languages*. James Cheney, ESOP 2008

- *Detection of Corrupted Schema Mappings in XML Data Integration* Dario Colazzo and Carlo Sartiani. ACM TOIT 2009.

W3C

- The let-rule:

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma, \$x : T \vdash Q2 : T}{\Gamma \vdash \text{let } \$x := Q1 \text{ return } Q2 : T}$$

- The for-rule:

$$\frac{\Gamma \vdash Q1 : T1 \qquad \Gamma \vdash \ \$x : \text{Prime}(T1) \rightarrow Q2 : T}{\Gamma \vdash \text{for } \$x \text{ in } Q1 \text{ return } Q2 : T.\ \text{Quant}(T1)}$$

- Examples:
  - Prime (a,b* | String)=a | b | String
  - Quant(a,b* | String)=+

Case-analysis vs W3C

XMark snippet:

```
<!ELEMENT site          (regions, categories, catgraph, people, .....)>
.....
.....
<!ELEMENT regions       (africa, asia, australia, europe, namerica,
                                  samerica)>
<!ELEMENT africa        (item*)>
<!ELEMENT asia          (item*)>
<!ELEMENT australia     (item*)>
<!ELEMENT namerica      (item*)>
<!ELEMENT samerica      (item*)>
<!ELEMENT europe        (item*)>
```

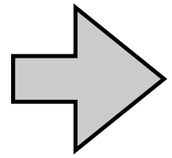
Query:

```
for $x /site/regions/node()
return <region>
                <name>name($x) </name>
                <total>count($x/item)</name>
        <region>
```

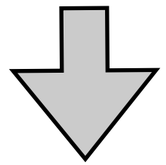
W3C query type:	region+
Case-analysis query type :	region,region,region,region,region,region

Inferring type-sets

bib → (book*)
book → (title, (author+ | editor+), publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String



```
for $x in /descendant :: author
return  $x/first, $x/last
```



{first, last}

Inferring type-sets

- Set of node types, instead of regular expressions, e.g.
- Can be easily obtained from previous systems
- Case-analysis and prime-quant operators are useless:
  - for-queries typed as let-queries
- Used for optimization purposes: next part.

TYPE BASED PROJECTION

XQuery implementations

- On top of RDBMSs, via shredding XML into tables and compiling XQuery into SQL
- Main-memory: tree-based representation kept in main-memory
  - Many implementations Galax, Saxon, Qizx, Zorba, BaseX, etc.
  - light-weight systems: easy to install, manage and integrate in a programming environment
  - quite efficient (indexing plus absence of disk I/O transfers)
  - Limitation: large documents cannot be processed.

XML projection

- Fact: queries tend to use a small part of the input document
- Optimization: to evaluate Q on D,
  - project D by pruning out sub-trees not needed by Q
  - evaluate Q on the projection D':

$$Q(D)=Q(D') \quad \text{and size}(D')<<\text{size}(D)$$

XML projection

- Fact: queries tend to use a small part of the input document
- Optimization: to evaluate Q on D,
  - project D by pruning out sub-trees not needed by Q
  - evaluate Q on the projection D':

$$Q(D)=Q(D') \quad \text{and size}(D')<<\text{size}(D)$$

- To make it feasible:
  - infer from Q information about its data-needs
  - use it to project D
  - keep memory usage as low as possible during projection

Path-based approach

- VLDB paper [MS03]: **path extraction** algorithm for XQuery
- Projection: D is streamed and matched against extracted paths
- Gains in terms of both memory and time
- Query engine unchanged
- Limitations:
  - projection-time depends on the number of extracted paths and //
  - only forward axes
  - low precision for queries like //*[a] (the path //* is extracted)

Schema-based approach

- VLDB paper [BCCN'06]: type-projector inference for XQuery.
- Use of schema information (DTD and XSD) for:
  - fast and buffer-less projection
  - dealing with backward axes
  - precise projection
- Main component: type system for statically inferring the type of data needed by the query (type-projector).

Intuition

data

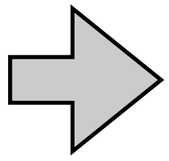
```
<bib>
   <book>
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
   </book>
   <book>
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
  </book>
   <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
  </book>
</bib>
```

schema

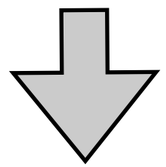
bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

Intuition

```
<bib>
   <book>
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
   </book>
   <book>
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
  </book>
   <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
  </book>
</bib>
```



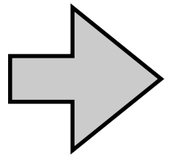
```
for $x in /bib/book/author
return  $x/first, $x/last
```



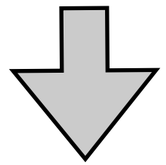
```
<first>W.</first> <last>Stevens</last>
<first>W.</first><last>Stevens</last>
```

Intuition

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String



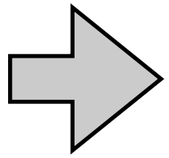
```
for $x in /bib/book/author
return  $x/first, $x/last
```



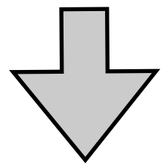
(first, last) *

Intuition

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String



```
for $x in /bib/book/author
return  $x/first, $x/last
```

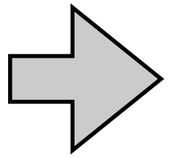


(first, last) *

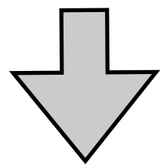
The **result** type describes query results

Intuition

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String



```
for $x in /bib/book/author
return  $x/first, $x/last
```



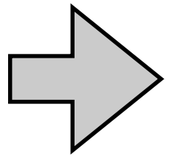
(first, last) *

The **result** type describes query results

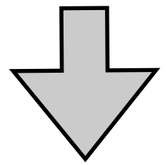
Types **matched/needed** to build the result type describe the query data needs

During Typing Process

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

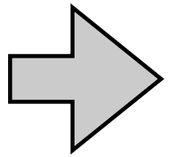


```
for $x in /bib/book/author
return  $x/first, $x/last
```

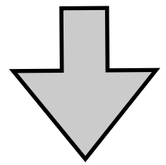


During Typing Process

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

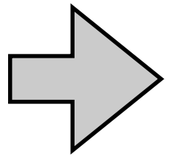


```
for $x in /bib/book/author
return  $x/first, $x/last
```

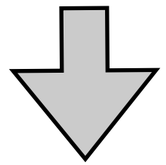


During Typing Process

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

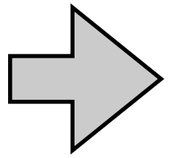


```
for $x in /bib/book/author
return  $x/first, $x/last
```

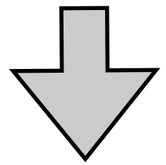


During Typing Process

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

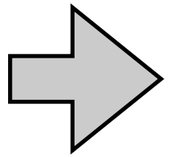


```
for $x in /bib/book/author
return  $x/first, $x/last
```

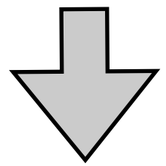


During Typing Process

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String



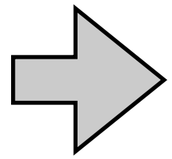
```
for $x in /bib/book/author
return  $x/first, $x/last
```



(first, last) *

During Typing Process

bib → (book*)
book → (title, (author+ |editor+) publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String



```
for $x in /bib/book/author
return  $x/first, $x/last
```

π={bib, book, author, first, last, String}

Type-projector: types matched/needed to build the result type

Type-based projection

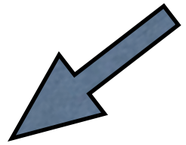
```
<bib>
   <book>
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
   </book>
   <book>
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
    <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>
</bib>
```

π={bib, book, author, first, last, String}

Type-based projection

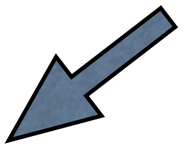
```
<bib>
   <book>
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
   </book>
   <book>
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
    <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>
</bib>
```

π={bib, book, author, first, last, String}



Type-based projection

```
<bib>
   <book>
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
   </book>
   <book>
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
    <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>
</bib>
```



π={bib, book, author, first, last, String}

Fast projection:
streaming & quasi-zero buffering

Type-based projection

```
<bib>
   <book>
      <title>TCP/IP Illustrated</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
   </book>
   <book>
      <title>Advanced Programming in the Unix environment</title>
      <author><last>Stevens</last><first>W.</first></author>
      <publisher>Addison-Wesley</publisher>
      <price>65.95</price>
   </book>
    <book>
      <title>The Economics of Technology and Content for Digital TV</title>
      <editor>
            <last>Gerbarg</last><first>Darcy</first>
             <affiliation>CITI</affiliation>
      </editor>
         <publisher>Kluwer Academic Publishers</publisher>
      <price>129.95</price>
   </book>
</bib>
```

π={bib, book, author, first, last, String}

Fast projection:
streaming & quasi-zero buffering

```
for $x in /bib/book/author
return  $x/first, $x/last
```

<first>W.</first> <last>Stevens</last>
<first>W.</first><last>Stevens</last>

Type-projector inference

- Low precision with standard type inference

  π_Q = return type + descendant of return types + intermediate types

- Q=//*/first, the type for //* covers the whole schema: useless projector.

- Type-based approach, two inference systems:
  - type inference system (all XPath axes)
  - projector inference system, based on type inference
- Focus on XPath expressions
- Generalization to FLWR via path extraction

Precise type inference

- Naif compositional approach: low precision
- //author/first/parent::node() has type {author, editor}
- Precise type is {author}
- Precise typing via use of contexts
  - infer a pair (T, CTX)
  - T is the inferred type
  - CTX includes all types traversed from the root to reach T types

bib → (book*)
book → (title, (author+ |editor+), publisher, price?
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

Precision via contexts

P=//author/last/parent::node()

Step	Type	CTX
//author	{author}	{bib, book, author}
/last	{last}	{bib, book, author, last}
/parent::node()		

bib → (book*)
book → (title, (author+ |editor+),
publisher, price
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

Precision via contexts

P=//author/last/parent::node()

Step	Type	CTX
//author	{author}	{bib, book, author}
/last	{last}	{bib, book, author, last}
/parent::node()	{author}	

{author, editor}∩ {bib, book, author,last}

bib → (book*)
book → (title, (author+ |editor+),
publisher, price
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

Precision via contexts

P=//author/last/parent::node()

Step	Type	CTX
//author	{author}	{bib, book, author}
/last	{last}	{bib, book, author, last}
/parent::node()	{author}	{bib, book, author}

bib → (book*)
book → (title, (author+ |editor+),
publisher, price
author → (last, first)
editor → (last, first, affiliation)
title → String
last → String
first → String
affiliation → String
publisher → String
price → String

Type-projector inference

Type-projector inference

- Type-projector is built upon inferred types and contexts.

Type-projector inference

- Type-projector is built upon inferred types and contexts.
- Inferred projector for //author/first includes $T_{//author} \cup CTX_{//author}$

Type-projector inference

- Type-projector is built upon inferred types and contexts.
- Inferred projector for //author/first includes $T_{//author} \cup CTX_{//author}$
- Inferred projector for //*/first does not include $T_{//*} \cup CTX_{//*}$

Type-projector inference

- Type-projector is built upon inferred types and contexts.
- Inferred projector for //author/first includes $T_{//author} \cup CTX_{//author}$
- Inferred projector for //*/first does not include $T_{//*} \cup CTX_{//*}$
  - $a \in T_{//*}$ contributes to the projector if a non-empty type is inferred for descendant::node/first starting from a

Type-projector inference

- Type-projector is built upon inferred types and contexts.
- Inferred projector for //author/first includes $T_{//author} \cup CTX_{//author}$
- Inferred projector for //*/first does not include $T_{//*} \cup CTX_{//*}$
  - $a \in T_{//*}$ contributes to the projector if a non-empty type is inferred for descendant::node/first starting from a
  - a is an ancestor of the first type

Type-projector inference

- Type-projector is built upon inferred types and contexts.
- Inferred projector for //author/first includes $T_{//author} \cup CTX_{//author}$
- Inferred projector for //*/first does not include $T_{//*} \cup CTX_{//*}$
  - $a \in T_{//*}$ contributes to the projector if a non-empty type is inferred for descendant::node/first starting from a
  - a is an ancestor of the first type
  - the projector includes {bib, book, author, editor}$\subset T_{//*}$

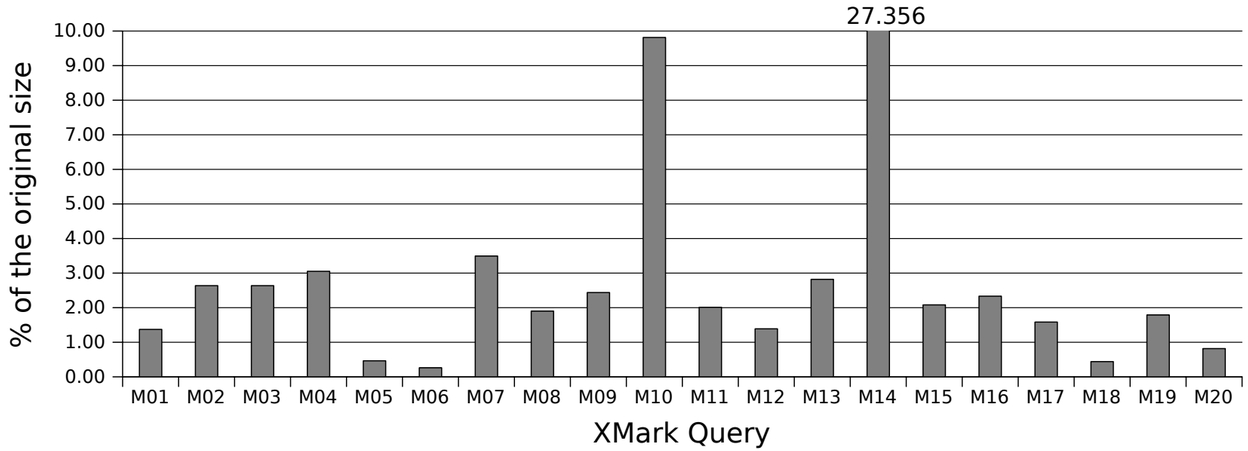
Other features

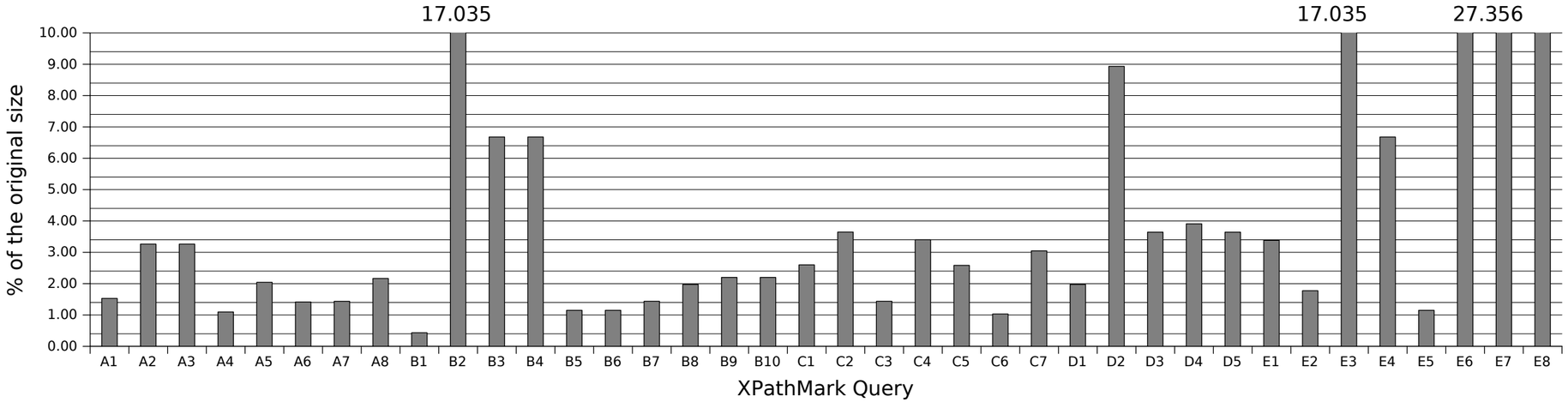
- Typing for paths with conditions
  - simple disjunctive form [P1 or ... or Pn]
  - conditions are approximated during path extraction

```
for $x in //*[author and price]
return $x/title          →   P1=//*[author or price]   P2=//*[author or price]/title//node()
```

- Horizontal axes:
  - following-sibling::NT rewritten to parent::node()/child::NT
  - this allows to encode the following axis (ancestor-or-self is needed)

Precision of type-projection





Type-projection for updates

Dealing with updates

- XQuery Update Facility (W3C)

U ::= delete Q | rename Q as a |

| insert Q as first/last into Q |

| replace Q as Q | U,U | for x in Q return U | .....

- Snapshot semantics: an update pending list (UPL) is determined and then applied to D.
- Recent works [FGP07, Far10] propose techniques to transform U into Q_U
- Q_U re-constructs all that is not updated, and returns new updated parts:
  - with query projection, the whole D is projected
- Projecting only for the UPL is challenging: need to touch the query engine.

Type-based projection for updates

- For queries we have $Q(D) = Q(D_\pi)$
- For updates: $U(D) \neq U(D_\pi)$
- Type-based approach: Merge process:
  - $Merge(D, U(D_\pi)) = U(D)$
  - D and $U(D_\pi)$ are visited in parallel, in a streaming fashion
  - no buffering: each step only depends on current D and $U(D_\pi)$ elements
  - U is not rewritten

Merge: soundness and no-buffering

- A new kind of type-projector, first extension:

  U= for $x in //book where $x/author/first='Dario' delete $x/price

- Low precision with π={bib, book, price, String}

- Bi-level projector π_{no}={bib, book, price} and π_{eb}={first}

- More precision: the type String is implicit

A new level for insert/replace

- $\pi_U=(\pi_{no}, \pi_{eb})$ is not enough to deal with insert/replace
- Merge needs more information for soundness and efficiency (no buffering)
- Consider

  U= for x in //book
  where (x/author and not x/price)
  insert <price >10</price> as last into x

- Tentative: bi-level projector π_{no}={bib,book,auhtor, price} and π_{eb}={}

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

U(D)

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
   <price>10<price>
  </book>
</bib>
```

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

U(D)

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
   <price>10<price>
  </book>
</bib>
```

same positions

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

U(D)

```
<bib>
```

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
   <price>10<price>
  </book>
</bib>
```

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

U(D)

```
<bib>
  <book>
```

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
    <price>10<price>
  </book>
</bib>
```

same positions

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

U(D)

```
<bib>
  <book>
```

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
    <price>10<price>
  </book>
</bib>
```

0 < 1 and title$\notin\pi$

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

U(D)

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>
```

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
    <price>10<price>
  </book>
</bib>
```

same positions

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

.....

U(D$_\pi$)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
    <price>10<price>
  </book>
</bib>
```

U(D)

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>
      <last >Stevens</last>
      <first >W.</first>
    </author>
```

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

?

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
    <price>10<price>
  </book>
</bib>
```

U(D)

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>
      <last >Stevens</last>
      <first >W.</first>
    </author>
```

Merging with π_{no}={bib,book,auhtor, price} and π_{eb}={}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

?

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0.'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
    <price>10<price>
  </book>
</bib>
```

U(D)

```
<bib>
  <book>
    <title>TCP/IP Illustrated</title>
    <author>
      <last >Stevens</last>
      <first >W.</first>
    </author>
```

We are stuck.
Not enough information from positions and π.

The one-level below component

U= for x in //book
where (x/author and not x/price)
insert <price >10</price> as last into x

The one-level below component

U= for x in //book
where (x/author and not x/price)
insert <price >10</price> as last into x

- A 3-levels projector: π_{no}= {bib,book,author,price} π_{eb}={} π_{olb}={book}

The one-level below component

U= for x in //book
where (x/author and not x/price)
insert <price >10</price> as last into x

- A 3-levels projector: π_{no}= {bib,book,author,price} π_{eb}={} π_{olb}={book}
- If a tag is in π_{olb} then children of nodes with this tag are projected.

The one-level below component

U= for x in //book
where (x/author and not x/price)
insert <price >10</price> as last into x

- A 3-levels projector: π_{no}= {bib,book,author,price} π_{eb}={} π_{olb}={book}
- If a tag is in π_{olb} then children of nodes with this tag are projected.
- All siblings of the inserted nodes are in the projection.

The one-level below component

U= for x in //book
where (x/author and not x/price)
insert <price >10</price> as last into x

- A 3-levels projector: π_{no}= {bib,book,author,price} π_{eb}={} π_{olb}={book}
- If a tag is in π_{olb} then children of nodes with this tag are projected.
- All siblings of the inserted nodes are in the projection.
- Small projections and enough information for correct synchronization

The one-level below component

U= for x in //book
where (x/author and not x/price)
insert <price >10</price> as last into x

- A 3-levels projector: π_{no}= {bib,book,author,price} π_{eb}={} π_{olb}={book}
- If a tag is in π_{olb} then children of nodes with this tag are projected.
- All siblings of the inserted nodes are in the projection.
- Small projections and enough information for correct synchronization

Merge with
π_{no}= {bib,book,author,price}, π_{eb}={} and π_{olb}={book}

D

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='0' >Stevens</last>
      <first pos='1' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
  </book>
</bib>
```

U(D)

```
<bib >
  <book>
    <title>TCP/IP Illustrated</title>
    <author>
      <last>Stevens</last>
      <first>W.</first>
    </author>
    <publisher>Addison-Wesley</publisher>
   <price>10<price>
  </book>
</bib>
```

U(D_π)

```
<bib pos='0'>
  <book pos='0' >
    <title pos='0'  >TCP/IP Illustrated</title>
    <author pos='1' >
      <last pos='2' >Stevens</last>
      <first pos='3' >W.</first>
    </author>
    <publisher pos='2' >Addison-Wesley</publisher>
   <price>10<price>
  </book>
</bib>
```

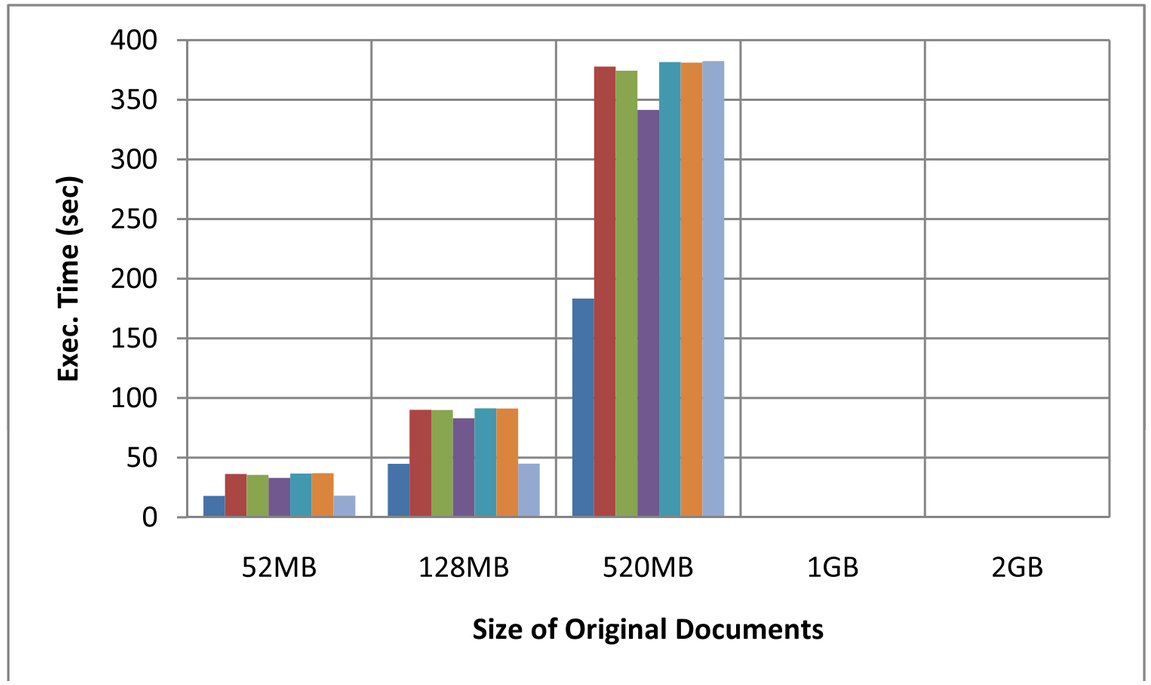
Inferring $\pi=(\pi_{no}, \pi_{eb}, \pi_{olb})$

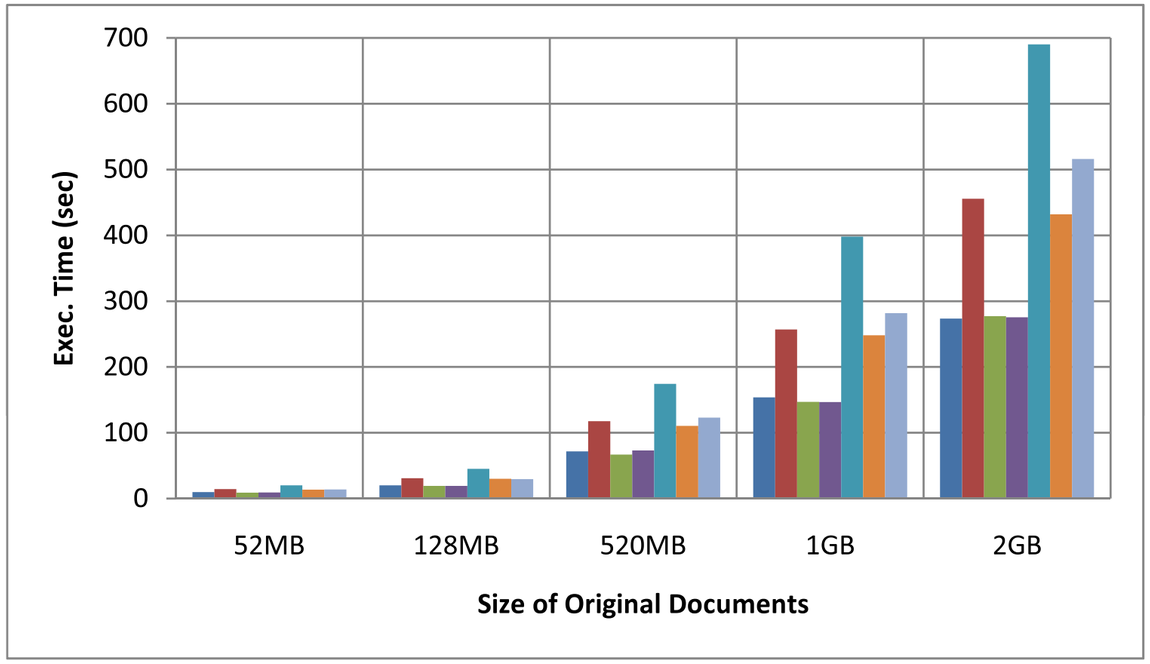
- The projector inference system for queries [BCCN'06] is used on paths extracted from U

- Path extraction rules need to classify paths according to the 3-level projector.

- Details in [BBCMS'11] and Amine Baazizi's Thesis.

Experiments: QuizX





Reduction of execution time as well.

TYPES FOR DETECTING QUERY-UPDATE INDEPENDENCE

Schema-less Independence

$$Q \perp U \quad \Leftrightarrow_{def} \quad \forall D.\ \ Q(U(D))=Q(D)$$

Schema-less Independence

$$Q \perp U \quad \Leftrightarrow_{def} \quad \forall D.\ \ Q(U(D))=Q(D)$$

- Crucial for efficient view maintenance under updates

Schema-less Independence

$$Q \perp U \quad \Leftrightarrow_{def} \quad \forall D.\ \ Q(U(D))=Q(D)$$

- Crucial for efficient view maintenance under updates
- Undecidable for the general case of XQuery and XUF [BC09]

Schema-less Independence

$$Q \perp U \quad \Leftrightarrow_{def} \quad \forall D.\ \ Q(U(D))=Q(D)$$

- Crucial for efficient view maintenance under updates
- Undecidable for the general case of XQuery and XUF [BC09]
- $Q \perp U$ iff $\forall D.$ U does not change Q used/return nodes in D

Schema-less Independence

$$Q \perp U \quad \Leftrightarrow_{def} \quad \forall D.\ \ Q(U(D))=Q(D)$$

- Crucial for efficient view maintenance under updates
- Undecidable for the general case of XQuery and XUF [BC09]
- $Q \perp U$ iff $\forall D$. U does not change Q used/return nodes in D
- Checking approaches: path extraction + XPath overlap checking :

Schema-less Independence

$$Q \perp U \quad \Leftrightarrow_{def} \quad \forall D.\ \ Q(U(D))=Q(D)$$

- Crucial for efficient view maintenance under updates
- Undecidable for the general case of XQuery and XUF [BC09]
- $Q \perp U$ iff $\forall D$. U does not change Q used/return nodes in D
- Checking approaches: path extraction + XPath overlap checking :
  - Read-delete/insert conflict: NP-complete for XPath[⇓,[], *], PTIME for XPath[⇓,*] [RS06]

Schema-less Independence

$$Q \perp U \quad \Leftrightarrow_{def} \quad \forall D.\ \ Q(U(D))=Q(D)$$

- Crucial for efficient view maintenance under updates
- Undecidable for the general case of XQuery and XUF [BC09]
- $Q \perp U$ iff $\forall D$. U does not change Q used/return nodes in D
- Checking approaches: path extraction + XPath overlap checking :
  - Read-delete/insert conflict: NP-complete for XPath[⇓,[], *], PTIME for XPath[⇓,*] [RS06]
  - Updates commutativity (similar problem) for XQuery updates [GRS09]: paths are extracted from 2 updates: these commutes if respective paths do not overlap

Path Based Analysis

- Q=for $x in $doc//country return ($x//name)
- U=for $x in $doc/country[population<25] return delete $x/city

query	accessed paths	updated paths
Q	$doc//country $doc//country//name//node()	
U	$doc/country $doc/country/population//node() $doc/country/city	$doc/country/city//node()

Path Based Analysis

- Q=for $x in $doc//country return ($x//name)
- U=for $x in $doc/country[population<25] return delete $x/city

query	accessed paths	updated paths
Q	$doc//country $doc//country//name//node()	
U	$doc/country $doc/country/population//node() $doc/country/city	$doc/country/city//node()

Path Based Analysis

- Q=for $x in $doc//country return ($x//name)
- U=for $x in $doc/country[population<25] return delete $x/city

query	accessed paths	updated paths
Q	$doc//country $doc//country//name//node()	
U	$doc/country $doc/country/population//node() $doc/country/city	$doc/country/city//node()

U impacts Q: city nodes deleted by U may have a country/name descendant, accessed/resulted by Q

More on Path-based Approaches [RS06] [GSR09]

- Pros: do not require schema information, efficient checking for large fragments of XQuery queries and updates

- Cons: no support for full XPath; if the schema is available, then low precision: dependence is risen when it should not

- The query //c and the update *delete* //b are considered as dependent, even if a schema S={r→a* ; a→b+,c*} is available

Schema-Based Independence Analysis for XML Updates [BC09]

U ::= delete Q | rename Q as a |

| insert Q as first/last into Q |

| replace Q as Q | U,U | ....

- **Impacted nodes (INs)** of U on D [BC09]: D nodes which are either a target of a rename or insert into command, or the parent of a target of a delete, replace update.

- By quoting [BC09]: *'Intuitively, the impacted nodes of an update are the nodes whose label or child sequence is changed by the update'.*

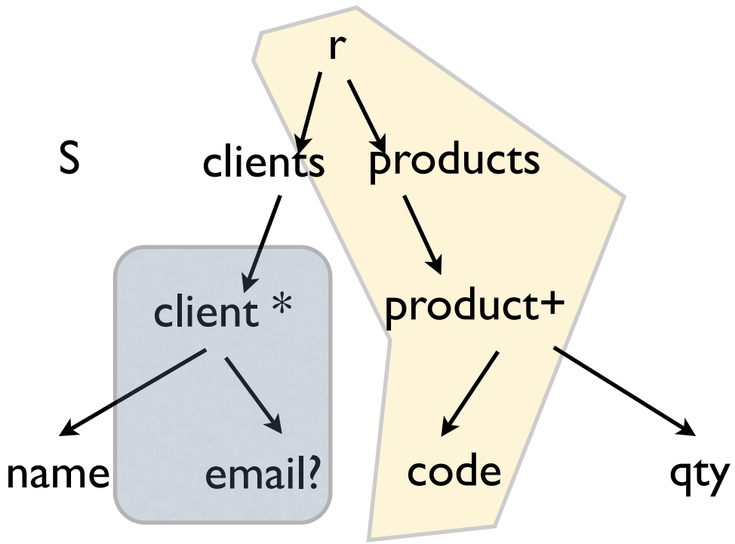
Schema-Based Independence Analysis for XML Updates [BC09]

- A schema S for queried/updated documents is available

$$Q \perp_S U \quad \Leftrightarrow_{def} \quad \forall D{:}S \quad Q(U(D))=Q(D)$$

- Type inference for inferring a *set* T_Q containing types of used and return nodes; similar to a type-projector.
- Type inference for inferring a *set* T_U containing types of IN_U nodes.
- Independence detection:
  - $T_U \cap T_Q = \varnothing$ then $Q \perp_S U$.

Example



Q= for $x in /r/products/product
return <co> $x/code </co>

U= delete //email

T_U={client} T_Q={r, products,product, code}

$T_U \cap T_Q = \varnothing$ hence $Q \perp_S U$

Some Test Results [BC09]

Upd#	t_r^{naive}	t_r^{ind}	t_c^{ind}	t_m^{ind}	Saved	Save%
U0	9.04	0.00	0.36	0.36	8.68	96%
UA1	8.90	7.32	0.39	7.71	1.19	13%
UA2	8.95	6.57	0.42	6.99	1.96	22%
UA3	8.93	6.53	0.40	6.93	2.00	22%
UA4	8.91	8.52	0.39	8.91	0.00	0%
UA5	8.94	8.65	0.39	9.04	-0.10	-1%
UA6	8.91	8.35	0.39	8.74	0.17	2%
UA7	8.88	8.50	0.39	8.89	-0.01	0%
UA8	8.95	8.60	0.39	8.99	-0.04	0%
UB1	8.90	8.59	0.38	8.97	-0.07	-1%
UB2	8.86	6.55	0.42	6.97	1.89	21%
UB3	7.89	6.06	0.39	6.45	1.44	18%
UB4	7.89	6.11	0.38	6.49	1.40	18%
UB5	8.92	8.49	0.39	8.88	0.04	0%
UB6	8.92	8.32	0.38	8.70	0.22	2%
UB7	8.96	8.39	0.41	8.80	0.16	2%
UB8	8.98	7.23	0.39	7.62	1.36	15%

Benefits of static independence checking are substantial (up to 22% for a 11MB document), and actually increase as the data size increases.

Schema-Based Independence Analysis for XML Updates [BC09]

- Pros: full XPath; polynomial technique, since both type sets T_Q and T_U can be inferred in polynomial time.

- Cons: precision could be improved in several cases, and in particular for those cases like //a//b and delete //c//b (when S says that b nodes can not have a and c ancestors at once).

Increasing precision [BCU12]

Increasing precision [BCU12]

- Inference of sets of chains of types

Increasing precision [BCU12]

- Inference of sets of chains of types
- For Q=//a//b and U=delete //c//b over {r→a,c ; a→b* ; c→b*}
  - query chain r.a.b
  - update chain r.c.b
  - non-conflicting chains, hence independence

Increasing precision [BCU12]

- Inference of sets of chains of types
- For Q=//a//b and U=delete //c//b over {r→a,c ; a→b* ; c→b*}
  - query chain r.a.b
  - update chain r.c.b
  - non-conflicting chains, hence independence
- Inferring chains is challenging:
  - infinite chains for recursive types,
  - exponential blowup even for non-recursive types

Increasing precision [BCU12]

- Inference of sets of chains of types
- For Q=//a//b and U=delete //c//b over {r→a,c ; a→b* ; c→b*}
  - query chain r.a.b
  - update chain r.c.b
  - non-conflicting chains, hence independence
- Inferring chains is challenging:
  - infinite chains for recursive types,
  - exponential blowup even for non-recursive types
- Solution: static analysis to upper-bound number of inferred chains, DAG representation for efficient representation of inferred chains.

Increasing precision [BCU12]

- Inference of sets of chains of types
- For Q=//a//b and U=delete //c//b over {r→a,c ; a→b* ; c→b*}
  - query chain r.a.b
  - update chain r.c.b
  - non-conflicting chains, hence independence
- Inferring chains is challenging:
  - infinite chains for recursive types,
  - exponential blowup even for non-recursive types
- Solution: static analysis to upper-bound number of inferred chains, DAG representation for efficient representation of inferred chains.
- Details in Federico's talk

Increasing precision [BCU12]

- Inference of sets of chains of types
- For Q=//a//b and U=delete //c//b over {r→a,c ; a→b* ; c→b*}
  - query chain r.a.b
  - update chain r.c.b
  - non-conflicting chains, hence independence
- Inferring chains is challenging:
  - infinite chains for recursive types,
  - exponential blowup even for non-recursive types
- Solution: static analysis to upper-bound number of inferred chains, DAG representation for efficient representation of inferred chains.
- Details in Federico's talk

Increasing precision [BCU12]

- Inference of sets of chains of types
- For Q=//a//b and U=delete //c//b over {r→a,c ; a→b* ; c→b*}
  - query chain r.a.b
  - update chain r.c.b
  - non-conflicting chains, hence independence
- Inferring chains is challenging:
  - infinite chains for recursive types,
  - exponential blowup even for non-recursive types
- Solution: static analysis to upper-bound number of inferred chains, DAG representation for efficient representation of inferred chains.
- Details in Federico's talk

ADVANCED TOPICS

Subclasses of Regular Expressions

- The complexity of decision problems is quite high
  - Inclusion (PSPACE-complete standard r.e.)
  - Intersection
- To efficiently solve these problems we need subclasses of regular expressions
  - Easy to define and to check
  - Not restrictive and capturing many practical cases

1-Unambiguity

- Not intuitive
- Mostly designed for easing membership checking
- Designed for regular expressions without counting and/or unordered concatenation/interleaving

RE	Membership	Inclusion	Intersection
RE	PTIME	PTIME	PSPACE [MNS04]
RE(#)	PTIME	coNP-hard [KT03]	PSPACE-hard [GGM09]

Strong Determinism

- Consider the following regular expression
  - (a[1..2] + b[1..2])[1..3]
- This expression is 1-unambiguous, as there is a single occurrence of each symbol
- However, when matching a word w we have multiple choices for matching the word
  - The word abb can be parsed
    - By unfolding the b counter twice and the outer counter once
    - Or by unfolding the b counter once and the outer counter twice.

Strong Determinism: Intuition

- Quoting [GGM09]
  - "For a strongly deterministic expression, we will additionally require that we always know how to go from one position to the next."
- Example
  - (a[0..1]·b[0..1])[0..2] is 1-unambiguous (each symbol appears once)
  - (a[0..1]·b[0..1])[0..2] is not strongly deterministic
    - After reading 'a', we can match 'b' in two ways

Bracketed Regular Expressions

- We follow [GGM09]
- We enrich each counter with an index: $s[k..l] \Rightarrow \langle_i s\rangle_i[k..l]$

- Given $\Gamma = \{\langle_i, \rangle_i \mid i \in N\}$, a regular expression on Σ is now on $\Sigma \cup \Gamma$
- Example
  - $(a[0..1] \cdot b[0..1])[0..2] \Rightarrow \langle_1(\langle_2 a\rangle_2[0..1]\ \langle_3 b\rangle_3[0..1])\rangle_1[0..2]$

Strongly Deterministic Regular Expressions - Formal Definition

- A string w in $\Sigma \cup \Gamma$ is *correctly bracketed* if w has no substring $\langle_i \rangle_i$
- Let $\tilde{r}$ be the bracketed expression obtained by r
- A regular expression with counting r is strongly deterministic if r is 1-unambiguous and there do not exist strings u,v,w over $\Sigma \cup \Gamma$, strings $\alpha \neq \beta$ over Γ, and a symbol $a \in \Sigma$ such that $u\alpha av$ and $u\beta aw$ are correctly bracketed in $L(\tilde{r})$

Examples

- Consider again
  - $(a[0..1] \cdot b[0..1])[0..2]$
    - and its bracketed counterpart $\langle_1(\langle_2 a\rangle_2[0..1]\ \langle_3 b\rangle_3[0..1])\rangle_1[0..2]$
- $\langle_1\langle_2 a\rangle_2\langle_3 b\rangle_3\rangle_1$ is correctly bracketed
- $\langle_1\langle_2 a\rangle_2\rangle_1\langle_1\langle_3 b\rangle_3\rangle_1$ is correctly bracketed too

Comments

- Strong determinism is a quite complex notion
- As for 1-unambiguity, no easy syntactical characterization
- A practical way to get strong determinism is to avoid counter nesting

Complexity of Decision Problems

Kind of RE	Membership	Inclusion	Intersection
RE w.d./s.d.	PTIME	PTIME [H10]	PSPACE [MNS04]
RE(#) w.d.	PTIME	coNP-hard [KT03]	PSPACE-hard [GGM09]
RE(#) s.d.	PTIME	in PSPACE [GGM09]	PSPACE [GGM09]

Chain Regular Expressions (CHARE)

- Conjunctive regular expressions of the form $f_1 \cdot \ldots \cdot f_n$ [MNS04]
- Each factor is a union of the form
  - $a_1 + \ldots + a_n$
  - $(a_1 + \ldots + a_n)?$
  - $(a_1 + \ldots + a_n)^+$
  - $(a_1 + \ldots + a_n)^*$
  - $a[k..l]$ $k \geq 0$

Examples

- a(b+c)*d+(e+f)? is a CHARE
- (ab + c)* is not a CHARE
- (a* + b?)* is not a CHARE

Comments

- A very simple syntactical restriction
  - Easy to understand
  - Easy to check
- CHAREs cover a large set of regular expressions used in practice [BNV04]
- CHAREs fail in significantly lowering the complexity of decisional problems

Complexity of Decisional Problems

Kind of RE	Membership	Inclusion	Intersection
CHARE	PTIME	PSPACE [MNS04]	PSPACE [MNS04]
CHARE(#)	PTIME	EXPSPACE [GMN07]	PSPACE [GMN07]
CHARE(a, a[k..l] k>0)	PTIME	PTIME [GMN07]	PTIME [GMN07]

Conflict Freedom

- A simple restriction for regular expressions with interleaving and counting
  - Each symbol may appear once
  - Counting is limited to symbols
- More formally:

$$r ::= \epsilon \quad | \quad a \quad | \quad r + r \quad | \quad r \cdot r \quad | \, r \& r \quad | \quad a[m..n]$$

$$r_1 \circledast r_2 \quad \Rightarrow \quad Sym(r_1) \cap Sym(r_2) = \emptyset$$

More on Conflict Freedom

- Expressions like (a+b)* can be modeled by a[0..*]&b[0..*]
- Conflict-free types are captured by a minimal set of constraints
- Membership is evaluated through constraint *derivatives*
  - Similar to Brzozowski derivatives [B64]
- Inclusion is reduced to constraint implication

Examples

- a[1..1]&(b[1..1] + c[1..1]) is conflict-free
- (a[1..1]&(b[1..1] + c[1..1]))[1..3] is not conflict-free

Complexity of Decisional Problems

Membership	Inclusion	Intersection
Linear [GCS08]	Quadratic [CGS09]	NP-hard [GCS07]

Comments

- A syntactical restriction
  - Easy to use
  - Easy to check
- Conflict-free regular expressions capture many practical use schemas
- Robust for asymmetrical inclusion [CGS09]
  - Inclusion is still quadratic if the left hand-side is not conflict-free

CONCLUSIONS

Some Open Problems

- Type inference for XML query/update languages in the presence of counting

- The complexity of inclusion for strongly deterministic regular expressions with counting

- The maximal subclass of regular expressions with interleaving and counting for which inclusion is polynomial

THANK YOU AND Q&A

References

[B64] J. Brzozowski. Derivatives of Regular Expressions. Journal of the ACM (JACM), Volume 11 Issue 4, Oct. 1964.

[BBCM+11] A. Baazizi, N. Bidoit-Tollu, D. Colazzo, N. Malla and M. Sahakyan. Projection for XML Update Optimization. International Conference on Extending Database Technology (EDBT), 2011.

[BC09] Michael Benedikt, James Cheney: Schema-Based Independence Analysis for XML Updates. PVLDB 2(1): 61-72 (2009)

[BCCN06]V. Benzaken, G. Castagna, D. Colazzo, and K. Nguyen: Type-based XML projection. VLDB, 2006.

References

[BCF03]V. Benzaken, G. Castagna, A. Frisch: CDuce: an XML-centric general-purpose language. ICFP 2003

[BKW98] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. Information and Computation, 142(2):182–206, 1998.

[BM04] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation 28 October 2004

[BNSV10]G. J. Bex, F. Neven, T. Schwentick, S. Vansummeren: Inference of concise regular expressions and DTDs. ACM Trans. Database Syst. 35 (2): (2010)

References

[BNV04] G.J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML schema: A practical study. In WebDB 2004, pages 79–84, 2004.

[BP+06] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan. Extensible Markup Language (XML) 1.1 (Second Edition). W3C Recommendation 16 August 2006.

[C08] James Cheney: Regular Expression Subtyping for XML Query and Update Languages. ESOP 2008

[C08a] James Cheney: FLUX: functional updates for XML. ICFP 2008: 3-14

References

[CGMS06]D. Colazzo, G. Ghelli, P. Manghi, and C. Sartiani: Static analysis for path correctness of XML queries. J. Funct. Program., 16(4-5), 2006.

[CGS09] Dario Colazzo, Giorgio Ghelli, Carlo Sartiani: Efficient asymmetric inclusion between regular expression types. ICDT 2009: 174-182

[CM01] James Clark and Makoto Murata. RELAX NG Specification. Committee Specification 3 December 2001

[CS09]Dario Colazzo and Carlo Sartiani. Detection of Corrupted Schema Mappings in XML Data Integration Systems. ACM Transactions on Internet Technology (TOIT), 2009.

[CS09]Dario Colazzo and Carlo Sartiani. Precision and Complexity of XQuery Type inference. ACM PPDP, 2011.

References

[DF+10]D. Draper, P. Fankhauser, M. Fernandez, A. Malhotra, K. Rose, M. Rys, J. Simeon, and P. Wadler. XQuery 1.0 and XPath 2.0 formal semantics. W3C Recommendation, December 2010.

[F05] M. Franceschet. XPathMark: an XPath benchmark for XMark generated data. International XML Database Symposium (XSYM), 2005.

[FCG04]W. Fan, C. Y. Chan, and M. Garofalakis. Secure XML Querying with Security Views. ACM SIGMOD Conference on Management of Data, 2004

[FSW01] Mary F. Fernández, Jérôme Siméon, Philip Wadler: A Semi-monad for Semi-structured Data. ICDT 2001

References

[FW04] David C. Fallside and Priscilla Walmsley. XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004

[GCS07] Giorgio Ghelli, Dario Colazzo, Carlo Sartiani: Efficient Inclusion for a Class of XML Types with Interleaving and Counting. DBPL 2007: 231-245

[GCS08] Giorgio Ghelli, Dario Colazzo, Carlo Sartiani: Linear time membership in a class of regular expressions with interleaving and counting. CIKM 2008: 389-398

[GGM09] Wouter Gelade, Marc Gyssens, Wim Martens: Regular Expressions with Counting: Weak versus Strong Determinism. MFCS 2009: 369-381

References

[GMN07] Wouter Gelade, Wim Martens, Frank Neven: Optimizing Schema Languages for XML: Numerical Constraints and Interleaving. ICDT 2007: 269-283

[GRS08] G. Ghelli, K. Rose, and J. Siméon: Commutativity analysis for XML updates. ACM Transactions on Database Systems, 33(4):1-47, 2008.

[H10] Dag Hovland: The Inclusion Problem for Regular Expressions. LATA 2010: 309-320

[HP+02] Mauricio A. Hernández, Lucian Popa, Yannis Velegrakis, Renée J. Miller, Felix Naumann, Ching-Tien Ho: Mapping XML and Relational Schemas with Clio. ICDE 2002: 498-499

References

[HP03]H. Hosoya, B. C. Pierce: XDuce: A statically typed XML processing language. ACM Trans. Internet Techn. 3(2): 117-148 (2003)

[K77] D. Kozen. Lower bounds for natural proof systems. In FOCS 1977, pages 254–266. IEEE, 1977.

[KS07] C. Koch, S. Scherzinger: Attribute grammars for scalable query processing on XML streams. VLDB J. 16(3): 317-342 (2007)

[KT03] Pekka Kilpeläinen, Rauno Tuhkanen: Regular Expressions with Numerical Occurrence Indicators - preliminary results. SPLST 2003: 163-173

References

[MNS04] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for simple regular expressions. In MFCS, pages 889–900, 2004.

[MS03]A. Marian and J. Siméon: Projecting XML documents. In VLDB, 2003.

[RS06]M. Raghavachari and O. Shmueli: Conflicting XML updates. In EDBT, 2006.

[SM04] C.M. Sperberg-McQueen. Notes on finite state automata with counters. http://www.w3.org/XML/2004/05/msm-cfa.html, 2004.

References

[SM73] L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time(preliminary report). In STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing, pages 1–9, New York, NY, USA, 1973. ACM Press.

[SW+01] Albrecht Schmidt, Florian Waas, Martin Kersten, Daniela Florescu, Ioana Manolescu, Michael J. Carey and Ralph Busse. The XML Benchmark Project. Technical Report. Centrum voor Wiskunde en Informatica, April, 2001.

[TB+04] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. XML Schema Part 1: Structures Second Edition. W3C Recommendation 28 October 2004

COMPLEXITY,
BASIC NOTIONS

Complexity

- A quick review of complexity
  - Time complexity
  - Space complexity
  - Complexity hierarchy and classes

$O(g(n))$ and $\Omega(g(n))$

- $f(n) \in O(g(n))$ if $f(n) \leq a \cdot g(n) + b$, where a and b are non-negative constants

- $f(n) \in \Omega(g(n))$ if $f(n) \geq a \cdot g(n) + b$, where a and b are non-negative constants

- $f(n) \in \Theta(g(n))$ if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$

Time Complexity

- DTIME(T(n)) is the class of problems that can be solved by a *deterministic* algorithm in time O(T(n))

- NTIME(T(n)) is the class of problems that can be solved by a *non-deterministic* algorithm in time O(T(n))

Space Complexity

- DSPACE(S(n)) is the class of problems that can be solved by a deterministic algorithm using at most O(S(n)) cells
- NSPACE(S(n)) is the class of problems that can be solved by a non-deterministic algorithm using at most O(S(n)) cells
- Cells can be
  - tape cells (Turing Machines)
  - memory cells (Random Access Machines)

Relation Between Time and Space

- DTIME(f(n)) ⊆ DSPACE(f(n))
- This means that, if your (deterministic) algorithm has polynomial time complexity, then it uses polynomial space
- The contrary is *not* true
- Your deterministic algorithm can perform an exponential number of steps on a polynomial space structure

DTIME vs NTIME

- DTIME(f(n)) ⊆ NTIME(f(n))
- DTIME(n) ⊊ NTIME(n)
- Strict containment applies only to O(n)

Complexity Classes 1/2

$$L = DSPACE(\log n)$$

$$NL = NSPACE(\log n)$$

$$P = U_{i \geq 1} DTIME(n^i)$$

$$NP = U_{i \geq 1} NTIME(n^i)$$

$$PSPACE = U_{i \geq 1} DSPACE(n^i)$$

$$NSPACE = U_{i \geq 1} NSPACE(n^i)$$

Complexity Classes 2/2

$$EXP = U_{i \geq 1} DTIME(2^{n^i})$$

$$NEXP = U_{i \geq 1} NTIME(2^{n^i})$$

$$EXPSPACE = U_{i \geq 1} DSPACE(2^{n^i})$$

$$NEXPSPACE = U_{i \geq 1} NSPACE(2^{n^i})$$

Co-classes

- Let C be a complexity class
- A problem P is in the complexity class co-C if its complement is in C
- All deterministic classes are closed under complement
  - EXPSPACE = co-EXPSPACE
  - PSPACE = co-PSPACE

C and co-C

- Non-deterministic classes closed under complement
  - NEXPSPACE = co-NEXPSPACE
  - NSPACE = co-NSPACE
  - NL = co-NL
- It is unknown whether the other non-determnistic classes are closed under complement
  - NP vs co-NP

Relations between complexity classes

- PSPACE = NSPACE
- EXPSPACE = NEXPSPACE
- $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
- $P \subset EXP \subset EXPSPACE$
- $PSPACE \subset EXPSPACE$
- $PSPACE \subseteq EXP$

Relations between complexity classes

- PSPACE = NSPACE
- EXPSPACE = NEXPSPACE
- $L \subseteq NL \subseteq P \subseteq NP \subseteq PSPACE$
- $P \subset EXP \subset EXPSPACE$
- $PSPACE \subset EXPSPACE$
- $PSPACE \subseteq EXP$

Hardness and completeness

- Given a complexity class C, a problem P is C-hard if every problem in C can be reduced to P in polynomial time

- Given a complexity class C, a problem P is C-complete if P is C-hard and $P \in C$

- For L and NL, we must use log-space reductions instead of poly-time reduction

- log-space reducibility implies polynomial time reducibility