Types for Correctness of Queries over Semistructured Data

Dario Colazzo

joint work with

Giorgio Ghelli, Paolo Manghi
and Carlo Sartiani

Università di Pisa
Overview

• Semistructured data
• XML representation
• Querying XML data: XQuery
• ∫ XQuery
• Typing
• Query correctness
• Our approach
Semistructured data

- Irregular and instable structure
- Self-describing representation
- Often, with no separate schema information (few guarantees of reliability and efficiency of applications)
- XML has become the standard for semistructured data representation
Semistructured data, examples

• BibTeX files
• Catalogs for products such as sensors: some sensors are rectangular and have height and width, while others are cylindrical and have diameter …. 
• Collections of data coming from several databases which may differ in structure
XML data, an example

<people>
   <person>
       <name>Dario Colazzo</name>
       <phone>+390002212765</phone>
   </person>
   <person>
       <name>Paolo Manghi</name>
       <tel>+390002212777</tel>
       <email>manghi@di.unipi.it</email>
   </person>
.............
</people>
XML Types: DTD, example

<!DOCTYPE people[
<!ELEMENT people (person+)
<!ELEMENT person (name, (phone | tel), email?)>
<!ELEMENT name #PCDATA>
<!ELEMENT phone #PCDATA>
<!ELEMENT tel #PCDATA>
<!ELEMENT email #PCDATA>]
}
Regular Expression Types

\[ E = \{
\begin{align*}
\text{AddressBook} & = \text{people} [\text{Person} +] \\
\text{Person} & = \text{person}[\text{Name}, \text{Phone}, \text{Email}] \\
\text{Name} & = \text{name}[\text{String}] \\
\text{Phone} & = \text{phone}[\text{String}] \mid \text{tel}[\text{String}] \\
\text{Email} & = \text{email}[\text{String}] \\
\end{align*}
\} \]
XQuery

- W3C XML query language
- Turing complete
- Based on XPath expressions
- Typed: result analysis
XQuery - examples

• Suppose d: AddressBook
• Two XQuery queries
  – d/person/name
  – d//name
• They can be respectively translated in the following equivalent ones:
  – for x in (d/person) return x child :: name
  – d self_or_descendant :: name
XQuery - examples

d/person

name
Dario.....

phone
+39 000050....

person

name
Paolo.....

tel
+39 000050....

email
manghi@....

MyThS, Bertinoro - Sept 2002
XQuery - examples

d//name

MyThS, Bertinoro - Sept 2002
XQuery: values

- **Trees**
  
  \[ t ::= \begin{array}{l} v_B \\ \vert \ l[f] \end{array} \]

- **Forests**
  
  \[ f ::= \begin{array}{l} () \\ \vert \ t \\ \vert \ f, f \end{array} \]
XQuery values, example

<person>
  <name>Dario</name>
  <phone>+390002212765</phone>
</person>

person[name=Dario, phone=+39 0002212765]
XQuery, types

- $T ::= ()$ empty sequence
- $B$ atomic type (char, int, …)
- $T | T$ union
- $T, T$ sequence
- $l[T]$ element type
- $X$ type name
Type environment

• Type environments: type definitions + type bindings for query free variables
  \[ E ::= () \]
  \[ X=T, E \]
  \[ x:X, E \]

• Hence:
  \[ T^* \quad X=() \mid T,X \]
  \[ T^+ \quad X= T \mid T,X \]
  \[ T? \quad X= () \mid T \]
Notation

• Given $s = \{x_1 = f_1, \ldots, x_n = f_n\}$

$$s :: E$$

means that

$$x_i = f_i \emptyset s \quad \text{iff} \quad x_i : T \emptyset E \text{ and } f_i \emptyset T$$

• $E \vdash Q$ means that each fv $x$ in $Q$ is typed in $E$, that is $x : T \emptyset E$
XQuery, syntax of queries

\[
Q ::= ()
\vert v_B
\vert x
\vert l[Q]
\vert Q,Q
\vert Q\text{ child::}l
\vert Q\text{ self_or_descendant ::}l
\vert \text{let } x:=Q \text{ return } Q
\vert \text{forsome } x \text{ in } Q \text{ return } Q
\vert \text{forall } x \text{ in } Q \text{ return } Q
\]
Query semantics

• Given a query \( Q \), a type environment \( E \) st \( E \vdash Q \), and \( s::E \), then

\[
[x]_s = s(x) \\
[Q_1, Q_2]_s = [Q_1]_s, [Q_2]_s \\
[Q \text{ child::l}]_s = f|l \quad \text{where} \quad [Q]_s = m[f]
\]

……..

For example

\[
(n[f_1], l[f_2], l[f_3])|l = l[f_2], l[f_3]
\]
Typing

• Query result analysis: does the query result match the expected output type?

• Query correctness: do the paths in the query make sense wrt the query input type?

• All the queries we have seen before are type correct.
Query correctness, observations

- Consider the simple type
  \[ X = l[a[Int] | b[Int]] \]

- For example, values \( l[a[1]] \) and \( l[b[2]] \) are both of type \( X \)

- Consider a query variable \( x:X \), then queries
  \[ Q_1 = x \text{ child} ::a \quad \text{and} \quad Q_2 = x \text{ child} ::b \]
  are both correct, and so is their product
  \[ Q_1, Q_2 = (x \text{ child} ::a), (x \text{ child} ::b) \]

MyThS, Bertinoro - Sept 2002
Correct queries

• An incorrect query
  \[ \text{x child::c} \]

• The path \text{child::c} does not make sense wrt the type of \text{x}

• Observe that
  \[ [\text{x child::c}]_s = ( ) \]
  for each
  \[ s :: (x:X, X = l[a[Int] | b[Int]]) \]
Correctness, definitions I

- One may find out that given Q and E st 
  E |-- Q, the query Q is correct iff 
  for each s::E [Q]_s \neq ()

- Wrong definition: non emptiness of query result is not equivalent to query correctness

- Just consider
  Q= (x child ::a), (x child::b), (x child ::c)
Correctness, definitions II

- A second possible solution could be to consider a value Error among other values, and to result it in the case of absence of matching:

  \[ [x \text{ child}::c]_s = \text{Error} \]

  for each s: (x:X, X=l[a[Int] | b[Int]] )

- Problem: error propagation!

- The query \( Q = (x \text{ child} :: a), (x \text{ child}::b) \) would result incorrect since for each substitution either (x child ::a) or (x child ::b) would result in Error
Correctness: our approach

- Consider the tree representation of Q, then we can focus on a subexpression (subtree) of Q by traversing a path \( \Box \) on Q.
- Define Paths(Q)\( \Box \) \{0,1\}* as the set of all possible path on Q; \( \Box \) is the empty path.
- Example: consider
  
  \[ Q = (x \text{ child ::a}), (x \text{ child::b}) \] , then

  \[ \text{Paths}(Q) = \{0,1, 0.0, 1.0\} \]
Correctness: our approach

- Given a query $Q$, a type environment $E$ such that $E|\|-- Q$, the query $Q$ is correct iff for each $\Box$ in $\text{Paths}(Q)$ there exists $s::E$
  
  $[Q]_s\Box$

- Intuitively, $[Q]_s\Box=\text{true}$ if, wrt the substitution $s$, then the local execution of $Q$ at path $\Box$ results in a non empty forest; otherwise, $[Q]_s\Box=\text{false}$.

- Example:
  
  $Q= (x \text{ child} ::a), (x \text{ child} ::b), (x \text{ child} ::c)$

  is not correct; for the path corresponding to the subquery $(x \text{ child} ::c)$ we always have the empty result.
Type System

- We have defined a set of type rules to check query correctness
- Rules are defined on judgements of the form $E |-- Q : (T, S)$
  where $T$ is the inferred type for the results of $Q$ and $S$ ranges over \{ok, error\}
- Soundness of the type system:
  - if $E |-- Q : (T, S)$ then $[Q] \sqsubseteq [T]$ and
    - $S=ok$ $\Rightarrow$ $Q$ is correct
    - $S=error$ $\Rightarrow$ $Q$ is not correct
Papers

• A first definition of the type system can be found in **A Typed Text Retrieval Query Language for XML Documents**, *Journal of the American Society for Information Science and Technology (JASIST)* Special Issue 2002.

• In **Types for Correctness of Queries over Semistructured Data**, WebDB2002 workshop, the system has been improved by a finer notion of query correctness and by the notion of path covering.