Regular XPath: Constraints, Query Containment and View-Based Answering for XML Documents

D. Calvanese\textsuperscript{1}, G. De Giacomo\textsuperscript{2}, M. Lenzerini\textsuperscript{2}, M.Y. Vardi\textsuperscript{3}

\textsuperscript{1} Free University of Bozen-Bolzano
\textsuperscript{2} Sapienza Università di Roma
\textsuperscript{3} Rice University

Logic in Databases (LID 2008)
Rome, Italy, 19-20 May 2008
Outline

1. Semistructured data and XML
2. Regular XPath
3. Reasoning over RXPath constraints using tree-automata
4. Implementation using symbolic techniques
5. RXPath queries
6. Conclusions
Semistructured data

According to [Abiteboul & Buneman & Suciu 2000]:

- Data not structured rigidly as in relational tables.
- Organized in terms of an edge and/or node labeled graph.
- Data is combined with schema information.
- Typically, some form of navigation is required to reach data items.
Semistructured data and XML

Nowadays the standard language for SSD is considered to be XML:
- Data represented as a node-labeled finite tree.
- The tree is unranked, i.e., the number of successors of a node is not bounded.
- Nodes have an identity, and may be referenced from other nodes.
- Siblings are ordered (document order).

Schema languages have been defined to constrain the structure of XML documents:
- DTDs (part of the W3C XML standard)
- XML Schema – adds data types
- Specialized DTDs [Papakonstantinou & Vianu 2000]
Characteristics of XPath

- W3C standard for querying and navigation of an XML tree.
- Provides the core for more powerful XML query and transformation languages (e.g., XQuery, XSLT).
- We view an XML document as a finite sibling-ordered tree:
  - Each node is labeled by a set of atomic propositions (that are an abstraction for XML elements, attributes, identifiers, ...).
  - Nodes are linked to each other by the relations child and right.
- XPath expressions can be interpreted in different ways:
  - To select a set of nodes.
  - To select a set of pairs of nodes.
  - To select a set of trees (the subtrees rooted at the selected nodes).
Syntax and semantics of XPath

XPath is a two-sorted language built over:

- an alphabet of atomic propositions $A$, each denoting a set of nodes,
- four atomic relations symbols (called steps):

$$S \rightarrow \text{child} \mid \text{parent} \mid \text{right} \mid \text{left}$$

Definition ((Core) XPath [Gottlob et al. 2002])

Consists of expressions of two sorts:

- path expressions, denoting sets of pairs of nodes:

$$P \rightarrow S \mid S^+ \mid \varphi? \mid P_1; P_2 \mid P_1 \cup P_2$$

- node expressions (a.k.a. filter expressions), denoting sets of nodes:

$$\varphi \rightarrow A \mid \langle P \rangle \mid \neg \varphi \mid \varphi_1 \land \varphi_2$$
Expressive power of XPath

**Theorem ([Marx’04])**

The answer sets of XPath expressions are exactly the sets definable in $\text{FO}^{\text{tree}}_2$, i.e., first-order queries over binary trees written using two variables only and using $S$ and $S^+$ as binary relations.

What if we want to relax the restriction of having only 2 variables in the FO query?

*Conditional XPath* adds conditional paths $(S; \varphi ?)^+$ to path expressions.

**Theorem ([Marx’05])**

The answer sets of Conditional XPath expressions are exactly the sets definable in $\text{FO}^{\text{tree}}$, i.e., first-order queries over binary trees using $S^+$ as binary relations.
Regular XPath root constraints

- Conditional XPath very much resembles Propositional Dynamic Logic (PDL) over trees.
- Here, we add some syntactic sugar and full transitive closure to obtain precisely PDL over finite sibling-ordered trees.

Definition (Regular XPath)

- We add an alphabet of identifiers $Id$, each denoting a single node.
- Path expressions, denoting sets of pairs of nodes:

  \[
  P \quad \rightarrow \quad f_{child} \mid \text{right} \mid P^{-} \mid \varphi? \mid P_1; P_2 \mid P_1 \cup P_2 \mid P^*
  \]

- Node expressions, denoting sets of nodes:

  \[
  \varphi \quad \rightarrow \quad A \mid Id \mid \langle P\rangle\varphi \mid [P]\varphi \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \varphi_1 \lor \varphi_2
  \]

We consider node expressions as constraints that should hold at the root of the tree.
Regular XPath vs. Conditional XPath

- **child** can be expressed as $fchild; right^*$. Hence, we can view (unranked) sibling trees as binary trees.
- $\langle P \rangle \varphi$ is a more convenient notation for $\langle P; \varphi? \rangle$.
- Over trees, we can express identifiers, by using a node formula $N_\alpha$ that forces proposition $\alpha$ to be true in exactly one node:

$$N_\alpha = \langle u \rangle \alpha \land$$
$$\left[ u \right]\left( (\langle fchild; u \rangle \alpha \rightarrow [\text{right}; u] \neg \alpha) \land \right.$$ 
$$\left( (\langle \text{right}; u \rangle \alpha \rightarrow [fchild; u] \neg \alpha) \land \right.$$ 
$$\left( \alpha \rightarrow ([fchild \cup \text{right}); u] \neg \alpha) \right))$$

Where $u$ is an abbreviation for $(fchild \cup \text{right})^*$. Hence, $u$ connects the root of the tree to every node.

*Note*: In PDL over arbitrary structures, this is not possible.

- Transitive closure $P^*$ of arbitrary path expressions $P$ increases the expressive power beyond that of $\text{FO}^{tree}$. 
Reasoning in Regular XPath

From the correspondence with Conditional XPath, we get:

**Theorem ([Marx’04])**

Satisfiability of an RXPath root constraint is $\text{ExpTIME}$-complete.

Caveat:

- The $\text{ExpTIME}$ upper bound is based on a reduction to reasoning in a variant of PDL.
- XML documents are finite trees. To express finiteness, we need to resort to repeat-CDPDL.
- Reasoning in repeat-CDPDL requires very heavy machinery (determinization construction by Safra and parity games) that up to now has resisted implementations.
We show how to encode satisfiability in RXPath into the problem of checking non-emptiness of an alternating two-way automaton on finite trees (2ATA).

We provide a non-emptiness algorithm for 2ATAs that can exploit symbolic techniques for dealing efficiently with a large number of states.
Two-way alternating automata on finite trees (2ATAs)

- A tree automaton accepts a set of finite node-labeled trees of a fixed out-degree. We can consider binary trees \((\text{fchild}, \text{right})\).
- Alternating automata generalize nondeterministic automata.
- Two-way automata can “run” up and down on the tree.

**Definition (2ATA on finite \(k\)-ary trees)**

Is a tuple \(A = \langle L, S, s_0, \delta \rangle\), with:

- \(L, S, s_0\) as for traditional finite automata.
- \(\delta : S \times L \rightarrow \mathcal{B}^+\left(\{-1, 0, 1, \ldots, k\} \times S\right)\)
- **Note**: there is no acceptance condition, i.e., the automaton accepts a tree if it finishes its run on the tree.
**Transition relation of 2ATAs**

**Intuition:** For each pair \((n', s')\) appearing in \(\delta(s, a)\), \(A\) spawns a copy of itself starting in state \(s'\) from the node specified by \(n'\).

**Example**

\[
\begin{align*}
\delta(s_0, \cdot) &= (0, s_a) \land (1, s_1) \\
\delta(s_1, b) &= (-1, s_a) \land ((1, s_c) \lor (2, s_d)) \\
\delta(s_a, a) &= \text{true} \\
\delta(s_a, b) &= \text{false}
\end{align*}
\]

... 

Accepts \(s_0 \rightarrow \varepsilon a\) and \(s_0 \rightarrow \varepsilon a\) but not \(s_0 \rightarrow \varepsilon a\).
Run of a 2ATA on a tree

A run of $A$ over a tree is again a tree with each node labeled by
- a node $x$ of the original tree, and
- a state $s$ of the 2ATA.

A node in the run labeled by $(x, s)$ describes a copy of $A$ that is in state $s$ and reads node $x$.

The run has to satisfy the transition function of $A$.

Example

Run for

$\delta(s_0, \cdot) = (1, s_1) \land (0, s_a) \land (2, s_2)$

$\delta(s_1, b) = (-1, s_a) \land ((1, s_c) \lor (2, s_d))$

on

```
  s_0 → ε → a
     ↑     ↓
     1     2
   11     12
   / \     / \
  a   b   a   d
```

Calvanese, De Giacomo, Lenzerini, Vardi
Regular XPath: Constraints over XML
2ATA for an RXPath root constraints

To check the satisfiability of an RXPath root constraint $\varphi$, we construct a 2ATA $A_\varphi$ for $\varphi$:

- 2ATA works on binary trees whose nodes are labeled with subsets of $\Sigma \cup \{ifc, hfc, irs, hrs\}$
  - $\Sigma$ is the alphabet of atomic propositions and identifiers.
  - $is/has$ first $child/right$ sibling are used for bookkeeping.

- We represent path expressions by NFAs (rather than REs) over
  $\{fchild, right, fchild^-, right^-\} \cup \{\psi? \mid \psi$ a node expression\}.

- The states of the 2ATA are essentially the sub-formulas of $\varphi$.

- The 2ATA inductively decomposes $\varphi$ while it traverses the tree and checks the constraints.
Inductive construction of the 2ATA

Transitions of 2ATA $A_\varphi = (\Lambda, S_\varphi, \varphi, \delta_\varphi)$ for $\varphi$:

- For an atomic symbol $\alpha \in \Sigma \cup \{ifc, irs, hfc, hrs\}$:
  \[\delta_\varphi(\alpha, \lambda) = \begin{cases} 
  \text{true}, & \text{if } \alpha \in \lambda \\
  \text{false}, & \text{if } \alpha \notin \lambda
  \end{cases}\]
  \[\delta_\varphi(\neg \alpha, \lambda) = \begin{cases} 
  \text{true}, & \text{if } \alpha \notin \lambda \\
  \text{false}, & \text{if } \alpha \in \lambda
  \end{cases}\]

- For the boolean:
  \[\delta_\varphi(\psi_1 \land \psi_2, \lambda) = (0, \psi_1) \land (0, \psi_2)\]
  \[\delta_\varphi(\psi_1 \lor \psi_2, \lambda) = (0, \psi_1) \lor (0, \psi_2)\]

- For $\langle P \rangle \psi$, where $P = (\Theta, Q, q_0, \varrho, F)$, there is a transition $\delta_\varphi(\langle P \rangle \psi, \lambda)$ that is the $\lor$ of:

  \begin{align*}
  \text{if } q_0 \in F & \text{ then } (0, \psi) \\
  \text{if } q \in \varrho(q_0, \text{fchild}) & \text{ then } (0, \text{hfc}) \land (1, \langle P_q \rangle \psi) \\
  \text{if } q \in \varrho(q_0, \text{right}) & \text{ then } (0, \text{hrs}) \land (2, \langle P_q \rangle \psi) \\
  \text{if } q \in \varrho(q_0, \text{fchild}^-) & \text{ then } (0, \text{ifc}) \land (-1, \langle P_q \rangle \psi) \\
  \text{if } q \in \varrho(q_0, \text{right}^-) & \text{ then } (0, \text{irs}) \land (-1, \langle P_q \rangle \psi) \\
  \text{if } q \in \varrho(q_0, \psi') & \text{ then } (0, \psi') \land (0, \langle P_q \rangle \psi)
  \end{align*}

- Similarly for $[P]\psi$. 
The 2ATA $A_\varphi$ assumes that the tree is well-formed:
- All bookkeeping information represented by $\{ifc, hfc, irs, hrs\}$ is in the right place.
- Identifiers are true in at most one node.

Well-formedness is ensured by a second 2ATA $A_{wf}$. For example:

\[
\begin{align*}
\delta(s_{ini}, \lambda) &= (0, \neg ifc) \land (0, \neg irs) \land (0, \neg hrs) \land \\
&\quad (0, s_{struc}) \land \\
&\quad \land_{Id \in \Sigma_{id}} (0, s_{Id}) \\
\delta(s_{struc}, \lambda) &= ((0, \neg hfc) \lor ((1, ifc) \land (1, \neg irs))) \land \\
&\quad ((0, \neg hrs) \lor ((2, irs) \land (2, \neg ifc))) \land \\
&\quad (1, s_{struc}) \land (2, s_{struc}) \\
&\quad \vdots
\end{align*}
\]
Complexity of RXPath satisfiability via 2ATAs

Size of $A_\varphi$ and $A_{wf}$:
- Alphabet is exponential in $|\varphi|$.
- Number of states is linear in $|\varphi|$.

Nonemptiness of a 2ATA with $n$ states and input alphabet of size $m$ can be decided in time exponential in $n$ and linear in $m$ (see later).

Theorem

Checking satisfiability of an RXPath root constraint by checking nonemptiness of the 2ATA $A_\varphi \cap A_{wf}$ can be done in ExpTime.
Nonemptiness of 2ATAs via symbolic methods

To check nonemptiness of a 2ATA, we convert it to an NTA:

- **NTAs have a simple emptiness algorithms:** compute bottom-up the set of accepting states, and check if initial state is in that set.

- **Problem:** the state space of the NTA is very large and cannot be stored explicitly.

- **Solution:** use symbolic techniques based on Reduced Ordered Binary Decision Diagrams (ROBDDs):
  - A BDD is a rooted DAG with two terminal nodes 0 and 1, each internal node labeled with a boolean variable and two outgoing edges labeled 0 and 1. Each node represents a boolean function.
  - In an OBDD, the variables are ordered, and paths visit them in the order.
  - In an ROBDD each node represents a distinct boolean function.

- **OBDDs allow for checking non-emptiness without constructing the NTA explicitly.**

- **Problem:** sets of states need to be represented symbolically, which is not possible for the NTA obtained by Safra’s construction.
From 2ATAs to NTAs

Observation:
- In a 2ATA, the structure of a run is very different from the structures of the tree underlying the run.
- In an NTA, the tree and the run have the same structure.

Ideas:
- Enrich the node labels so that they encode the choices of the 2ATA that satisfy the formula in the transition.
  \( \sim \) **Strategy tree**: each node is labeled with a set of triples \((s, k, s')\).
- To deal with paths that go up and down, annotate the nodes of the strategy tree with a directed graph on the states of the 2ATA, that encodes information about finite paths.

**Theorem**

A 2ATA \( \mathcal{A} \) accepts a tree \( T \) iff \( \mathcal{A} \) has a strategy tree \( \tau \) on \( T \) and an accepting annotation of \( \tau \).
From 2ATAs to NTAs (cont’d)

Theorem

Given a 2ATA $A$ with states $S$, there is an equivalent NTA $A_n$ whose states are pairs consisting of subsets of $S$ and $S \times S$.

- The set of states of $A_n$ can be described by a boolean function on $S^3$.
- The transition function of $A_n$ can also be described by a boolean function.

$\leadsto$ We can represent such function using ROBDDs and use symbolic techniques for checking nonemptiness.
RXPath queries

Definition

An RXPath query $Q$ is an RXPath path expressions returning the set of pairs of nodes that satisfy the expression.

We can polynomially reduce inference problems on RXPath queries to satisfiability of RXPath root constraint:

- **Satisfiability** of $Q$ under a set $\Gamma$ of RXPath root constraints:
  
  $Q$ satisfiable under $\Gamma$ iff $\Gamma \cup \{\langle u; Q \rangle \text{true} \}$ is satisfiable

- **Query containment** under a set $\Gamma$ of RXPath root constraints:
  
  $\Gamma \models Q_1 \subseteq Q_2$ iff
  
  $\Gamma \cup \{\langle u; Id_{st}?; Q_1 \rangle Id_{end}, [u; Id_{st}?; Q_2]\neg Id_{end} \} \text{unsat}$

- **View-based query answering** can also be reduced to satisfiability.
Conclusions

- This work is an example of Leonid’s argument of cross fertilization between verification and DBs.
- The optimization techniques we sketched are inspired by analogous techniques developed in verification.
- More generally, we expect that the DB community will be able to leverage implementations from the recent dramatic improvements in verification.