Towards Schema-Guided XML Query Induction

Jérôme Champavère    Rémi Gilleron
Aurélien Lemay    Joachim Niehren

Université de Lille – INRIA, France

ICML-2007 Workshop on
Challenges and Applications of Grammar Induction
June 24, 2007
Query on XML Documents

Famous Oregonians

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cecil D. Andrus</td>
<td>secretary of the interior</td>
</tr>
<tr>
<td>James Beard</td>
<td>food expert</td>
</tr>
<tr>
<td>Raymond Carver</td>
<td>writer, poet</td>
</tr>
<tr>
<td>Homer Davenport</td>
<td>political cartoonist</td>
</tr>
<tr>
<td>Douglas Engelbart</td>
<td>inventor</td>
</tr>
<tr>
<td>Matt Groening</td>
<td>cartoonist</td>
</tr>
<tr>
<td>Mark Hatfield</td>
<td>senator</td>
</tr>
<tr>
<td>Joni Huntley</td>
<td>track athlete</td>
</tr>
<tr>
<td>David Kennerly</td>
<td>photographer</td>
</tr>
</tbody>
</table>
Query on XML Documents

The query is to extract the name-function pair list of this web page.
The query is to extract the name-function pair list of this web page. The result can be put in a regular database.
Tree View for Queries

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>L. Pauling</td>
<td>Chemist</td>
</tr>
<tr>
<td>M. Groening</td>
<td>Cartoonist</td>
</tr>
</tbody>
</table>
**Tree View for Queries**

### Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>L. Pauling</td>
<td>Chemist</td>
</tr>
<tr>
<td>M. Groening</td>
<td>Cartoonist</td>
</tr>
</tbody>
</table>

**Diagram:**

```
TABLE

TR

<table>
<thead>
<tr>
<th>TH</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>L. Pauling</td>
</tr>
<tr>
<td>Function</td>
<td>Chemist</td>
</tr>
</tbody>
</table>

TR

<table>
<thead>
<tr>
<th>TH</th>
<th>TD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>M. Groening</td>
</tr>
<tr>
<td>Function</td>
<td>Cartoonist</td>
</tr>
</tbody>
</table>
```
Tree View for Queries

<table>
<thead>
<tr>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>L. Pauling</td>
<td>Chemist</td>
</tr>
<tr>
<td>M. Groening</td>
<td>Cartoonist</td>
</tr>
</tbody>
</table>
Query Induction

Representation of Queries

- Programs: C, Perl, ...
- Ad hoc formalisms: XPath, XQuery, ...
- Other formalisms: relational algebra, MSO formula, ...

Conception of Queries

- Hand-made: hard, error-prone, need for expert
- With visual tools: like Lixto [Gottolb, Koch 2000], still not easy
- Automatically induced: users just give some examples of wanted information
Query Induction

Representation of Queries
- Programs: C, Perl, ...
- Ad hoc formalisms: XPath, XQuery, ...
- Other formalisms: relational algebra, MSO formula, ...

Conception of Queries
- Hand-made: hard, error-prone, need for expert
- With visual tools: like Lixto [Gottolb, Koch 2000], still not easy
- Automatically induced: users just give some examples of wanted information

Our approach: Queries are represented by tree automata, and inferred by an RPNI-like algorithm.
Schema-guided Induction

The learning task faces some challenges.

- Very few examples obtained from users
- All available information has to be used
- The schema is such information (DTD, XML Schema, Relax NG, ...)

We investigate here how this information can be used for query induction.
1 Queries over XML Documents
2 Schema-guided Learning
3 Efficient Inclusion Checking
4 Learning from Partially Annotated Examples
Outline

1. Queries over XML Documents
2. Schema-guided Learning
3. Efficient Inclusion Checking
4. Learning from Partially Annotated Examples
Query as Tree Language

Consider the monadic query extracting names. Two equivalent views:

- Function from trees to annotated trees
- Language of correctly annotated trees

This language is functional.
Node Selecting Tree Transducers (NSTT Ts)

Queries represented by tree automata (regular queries).

- Represented here as string automata
- In fact, classical tree automata operating on Curried encoding
Queries over XML Documents

Node Selecting Tree Transducers (NSTTs)

Queries represented by tree automata (regular queries).

- Represented here as string automata
- In fact, classical tree automata operating on Curried encoding
Node Selecting Tree Transducers (NSTTts)

Queries represented by tree automata (regular queries).

- Represented here as string automata
- In fact, classical tree automata operating on Curried encoding
Answering Queries

- Find the proper boolean annotation
- At most one possible annotation (functionality of queries)
Answering Queries

- Find the proper boolean annotation
- At most one possible annotation (functionality of queries)

Query answering: $O(|t| \times |A|)$
How To Learn Queries

- NSTTs are tree automata
- Can be learned using RPNI [Oncina & García, 1993]
- One difference: no negative examples, use of functionality instead
Query Induction Issues

- Theoretical results
  - Theoretical characteristic sample: more than 1000 trees
  - Availability: about 3 trees

- In practice
  - Works on monadic queries with heuristics
  - More complicated for $n$-ary queries

- Main difficulty: very few examples, leads to wrong generalizations
  - Generalizations out of the domain
  - Use of domain knowledge allows to avoid them
Outline

1 Queries over XML Documents
2 Schema-guided Learning
3 Efficient Inclusion Checking
4 Learning from Partially Annotated Examples
Schemas: languages describing valid documents
- For XML: **DTD**, XSD, Relax NG
- HTML has a DTD
- Subclasses of regular tree languages
Schemas: languages describing valid documents

For XML: **DTD**, XSD, Relax NG

HTML has a DTD

Subclasses of regular tree languages

Queries can be encompassed by schemas represented by tree automata.
Origin of Schemas

Schemas can be either given or inferred:

- **Given:** DTD of HTML typically
- **Inferred:** DTD inference algorithm
  - Advantage: more precise than the DTD of HTML
  - But more difficult to obtain
Using Schema

Typing bias [Coste et al., 2004]

- States are typed according to a typing automaton from the domain
- Only states of the same type are merged
- Goal: constrain the structure of inferred automata

For us, the form of the automaton doesn’t matter, whereas language does.

Inclusion

We use inclusion into schema to prevent wrong generalizations.

- Inferred automata should satisfy the DTD at each step
- Domain bias [Coste et al., 2004]
  - Use of complement as negative examples
  - Promising experimental results
Learning with Inclusion Checking

Schema-guided RPNI

**Input:** Sample and schema
  - Run RPNI as usual and test inclusion at each step

**Output:** NSTT
Learning with Inclusion Checking

Schema-guided RPNI

**Input:** Sample and schema
- Run RPNI as usual and test inclusion at each step

**Output:** NSTT

Need for an efficient inclusion test.
Outline

1. Queries over XML Documents
2. Schema-guided Learning
3. Efficient Inclusion Checking
4. Learning from Partially Annotated Examples
Inclusion Test Principle

- Deterministic DTD over $\Sigma$
- Linear transformation into deterministic tree automaton $D$
  [Brüggemann-Klein & Wood, 1998]

```
<!ELEMENT TABLE (TR*) >
<!ELEMENT TR (TH, TH| TD, TD)>      
<!ELEMENT TH (#PCDATA)>            
<!ELEMENT TD (#PCDATA)>            
```

- Current hypothesis NSTT over $\Sigma \times \text{Bool}$ (i.e. annotated)
- Projection onto $\Sigma$: tree automaton $A$

$$L(A) \subseteq L(D) \iff \nexists \ t \in L(A) \cap L(D)^C$$
Efficient Inclusion Checking

Classical Method

- Complete $D$
- Compute $D^C$
- Compute $A \times D^C$
- Check that $A \times D^C$ has no accessible final state.

$$L(A) \subseteq L(D) \iff L(A \times D^C) = \emptyset$$
Efficient Inclusion Checking

Classical Method

- Complete $D$
- Compute $D^C$
- Compute $A \times D^C$
- Check that $A \times D^C$ has no accessible final state.

$$L(A) \subseteq L(D) \iff L(A \times D^C) = \emptyset$$

Time complexity in $O(|A| \times |\Sigma| \times |D|^2)$ because of complement of $D$ computation, which needs $D$ to be complete.
Our Approach for Testing Inclusion

- Avoid the complement of $D$ computation
- Accessibility in $A \times D$ instead of $A \times D^C$
- Detection of one inclusion failure:
  - Failure 1: problem with initial rules
  - Failure 2: problem with internal rules
  - Failure 3: problem with final states

**Proposition:** Inclusion $L(A) \subseteq L(D)$ fails if and only if one of the tree failure conditions occurs.
Failure 1

No TH in the DTD: automaton $A$ has a rule $TH \rightarrow 3$ whereas automaton $D$ has no state $q$ such that $TH \rightarrow q$.

Time complexity in $O(|\text{states}(A)| \times |\text{states}(D)|)$.
Failure 3

Trees evaluated in \((5, d)\) by \(A \times D\) with 5 final in \(A\) but \(d\) not final in \(D\).

Time complexity in \(O(|A| \times |D|)\), i.e. time to compute accessible states of \(A \times D\).
Failure 2

- Accessible states \((3, c)\) and \((2, a)\)
- Rule \(3 \rightarrow 4\) in \(A\)
- No rule \(c \rightarrow X\) in \(D\)

We look for such configurations during computation of accessible states of \(A \times D\), without enumerating all states of \(A \times D\).
Complexity Considerations

Failure 2

- Naive approach in $O(|A|^2 \times |D|^2)$
- Algorithmic trick leads to $O(|A| \times |D|)$
Complexity Considerations

Failure 2

- Naive approach in $O(|A|^2 \times |D|^2)$
- Algorithmic trick leads to $O(|A| \times |D|)$

**Theorem:** Inclusion $L(A) \subseteq L(D)$ can be computed in $O(|A| \times |D|)$ when $D$ is deterministic.

**Corollary:** Runtime of schema-guided RPNI is $O(|S|^3 \times |D|)$ where $S$ is a sample of positive examples and $D$ a schema.
Outline

1 Queries over XML Documents

2 Schema-guided Learning

3 Efficient Inclusion Checking

4 Learning from Partially Annotated Examples
Limitations of Completely Annotated Examples

Until now, input: *completely* annotated documents.

Two main problems:
- Time consuming
Limitations of Completely Annotated Examples

Until now, input: *completely* annotated documents.

Two main problems:

- Time consuming
Limitations of Completely Annotated Examples

Until now, input: \textit{completely} annotated documents.

Two main problems:

- Time consuming
- Whole document considered
Limitations of Completely Annotated Examples

Until now, input: *completely* annotated documents.

Two main problems:
- Time consuming
- Whole document considered

Solution: pruning trees
[Carme et al., 2007]
Principle of Pruning

- **Input:** set of **partially annotated** trees.
**Principle of Pruning**

- **Input:** set of *partially annotated* trees.
- **Our strategy:** We prune subtrees not on a path of selected nodes as they are *irrelevant* or *unannotated.*
**Principle of Pruning**

- **Input:** set of *partially annotated* trees.
- **Our strategy:** We prune subtrees not on a path of selected nodes as they are *irrelevant* or *unannotated* and annotate negatively the rest.
**Principle of Pruning**

- **Input:** set of *partially annotated* trees.
- **Our strategy:** We prune subtrees not on a path of selected nodes as they are *irrelevant* or *unannotated* and annotate negatively the rest.

![Diagram of a tree with nodes labeled TABLE, TD+, TD-, TR-, T, and T.]

- We obtain *completely annotated* pruned trees.
Learning with Pruning

- Using RPNI: First we prune, then we learn
- No functionality anymore, but *cut-functionality*
- **Output:** NSTT recognizing completely annotated pruned tree languages
Answering Queries Using Pruned NSTT

- Find consistent pruned trees
- $\top$ can be replaced by any subtree
- Answer in $O(|t| \times |A|)$
Learning from Partially Annotated Examples

Answering Queries Using Pruned NSTT

- Find consistent pruned trees
- $\top$ can be replaced by any subtree
- Answer in $O(|t| \times |A|)$
Learning from Partially Annotated Examples

Answering Queries Using Pruned NSTT

- Find consistent pruned trees
- $\top$ can be replaced by any subtree
- Answer in $O(|t| \times |A|)$
Answering Queries Using Pruned NSTT

- Find consistent pruned trees
- $\top$ can be replaced by any subtree
- Answer in $O(|t| \times |A|)$
Answering Queries Using Pruned NSTT

- Find consistent pruned trees
- $\top$ can be replaced by any subtree
- Answer in $O(|t| \times |A|)$
Flaws of Pruned NSTT

- The previous pruning technique cannot be used with schema inclusion.
- This tree for instance can be annotated:
Flaws of Pruned NSTT

- The previous pruning technique cannot be used with schema inclusion
- This tree for instance can be annotated:
Flaws of Pruned NSTT

- The previous pruning technique cannot be used with schema inclusion.
- This tree for instance can be annotated:

```
TABLE−
  TR−
    TR−
      TH−
    TR−
      TH−
    TD−
  TR−
    TR−
      TH−
    TD−

⊤ TR−
  TR−
    TR−
      TD−
    TD−
```

- But it doesn’t satisfy the DTD!
Schema-guided Pruning

Trees are pruned using states of the DTD instead of $\top$.
Schema-guided Pruning

Trees are pruned using states of the DTD instead of \( \top \).
Schema-guided Pruning

Trees are pruned using states of the DTD instead of $\top$. 

\[
\begin{array}{c}
\text{TABLE} \\
\text{TR} \\
\text{TD} \\
\text{TH}
\end{array}
\]
Schema-guided Pruning

Trees are pruned using states of the DTD instead of $\top$. 

![Diagram of pruned tree structure]
Query Answering with Schema-guided Pruning

Same as before, except that states are replaced by consistent subtrees.

Match.

Do not match.
Schema-guided Learning with Pruning

**Input:** Set of partially annotated trees and schema
- Prune the trees w.r.t. the schema
- Run RPNI with inclusion test

**Output:** NSTT operating on trees pruned w.r.t. the schema
Conclusion

Summary

- Use of inclusion test in query induction
- New efficient inclusion checking algorithm
- Schema-guided pruning

Work in Progress

- Experiments
- Adaptation to $n$-ary queries (especially pruning)
References


