

# Query Reasoning on Trees with Types, Interleaving, and Counting

**Pierre Genevès**<sup>1</sup>

joint work with Everardo Bárcenas<sup>2</sup>, Nabil Layaïda<sup>2</sup>, and Alan Schmitt<sup>2</sup>

<sup>1</sup> CNRS      <sup>2</sup> INRIA  
France

July 21<sup>st</sup>, 2011  
International Conference on Artificial Intelligence  
Barcelona, Spain

# Query Reasoning in the Context of XML

## Documents

Abstracted by finite trees

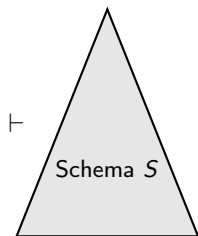
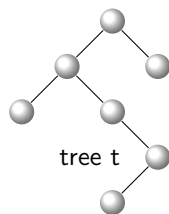
## Schema

- A Schema defines a class of documents (a tree language)
- XML Schema, DTD, Relax NG...
- Regular expressions restrict admissible content

## Programs that manipulate documents

- Queries navigate in the tree for selecting nodes and extracting information
- Queries are path expressions (à la XPath)

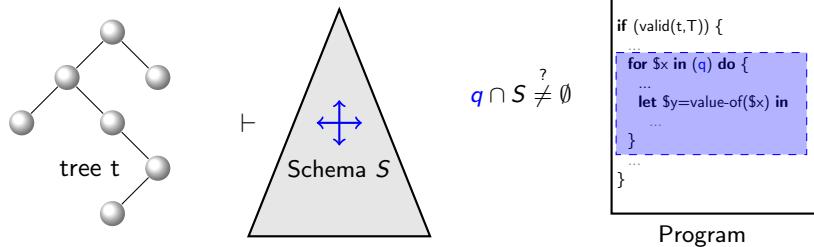
# Query Reasoning Problems and Applications



```
if (valid(t,T)) {  
  ...  
  for $x in (q) do {  
    ...  
    let $y=value-of($x) in  
    ...  
  }  
  ...  
}
```

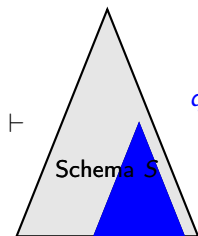
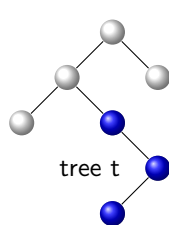
Program

# Query Reasoning Problems and Applications

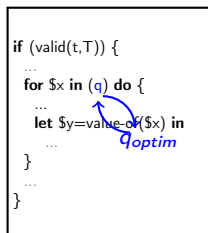


- Query satisfiability: detecting errors (inconsistent queries)
- Query containment/equivalence: proving optimizations (preserving semantics)
- Query intersection: checking program properties (security holes)

# Query Reasoning Problems and Applications



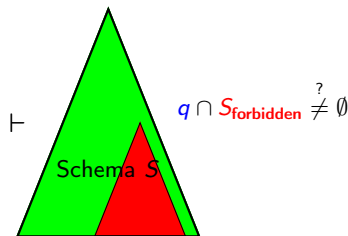
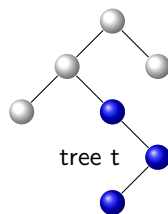
$$q \cap S \stackrel{?}{\equiv} q_{optim}$$



Program

- Query satisfiability: detecting errors (inconsistent queries)
- Query containment/equivalence: proving optimizations (preserving semantics)
- Query intersection: checking program properties (security holes)

# Query Reasoning Problems and Applications

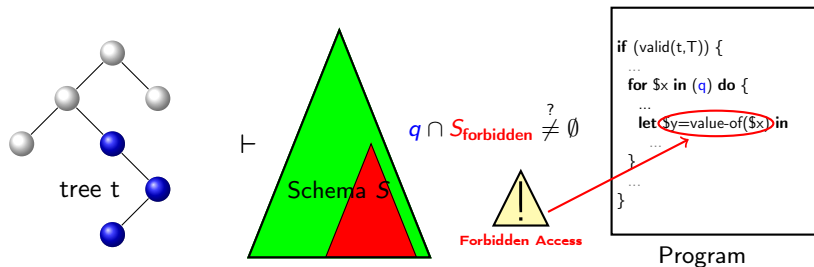


```
if (valid(t,T)) {  
  ...  
  for $x in (q) do {  
    ...  
    let $y=value-of($x) in  
    ...  
  }  
  ...  
}
```

Program

- Query satisfiability: detecting errors (inconsistent queries)
- Query containment/equivalence: proving optimizations (preserving semantics)
- Query intersection: checking program properties (security holes)

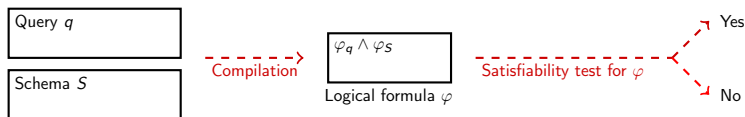
# Query Reasoning Problems and Applications



- Query satisfiability: detecting errors (inconsistent queries)
- Query containment/equivalence: proving optimizations (preserving semantics)
- Query intersection: checking program properties (security holes)

# Existing XPath and Schema Static Analyzers

Based on a reduction to logical satisfiability, 2 steps:



Approach proposed in [Geneves-PLDI'07,ICFP'09,WWW'10,ICFP'11]

- The logic:  $\mu$ -calculus for finite trees
- Supported query language: navigational XPath
- Supported schema language: regular tree grammars
- Combined complexity: EXPTIME (linear translations + sat. in  $2^{\mathcal{O}(n)}$ )
- Implementation: <http://wam.inrialpes.fr/websolver>

# Going Further: Challenges

## Several directions

- Growing logical **expressive power**? (currently MSO)
- Decreasing combined **complexity**? (impossible without dropping features: containment for regular tree grammars is hard for EXPTIME)
- Augmenting **succinctness** of the logic → good potential

## Succinctness is crucial

- A blow-up in the logical translations affects the combined complexity
- Augmenting succinctness is a way to address more problems in EXPTIME

# The Interleave Operator: Example

## Defining the class of documents for Multiple Choice Questions (MCQs)

- Each question has 1 correct answer, 1 incorrect answer, 1 irrelevant answer
- Order of answers must not matter (randomly generated)

Schema for validating a MCQ:

```
question[ (correctAnswer, incorrectAnswer, irrelevantAnswer) | (correctAnswer, irrelevantAnswer, incorrectAnswer)
          | (incorrectAnswer, correctAnswer, irrelevantAnswer) | (incorrectAnswer, irrelevantAnswer, correctAnswer)
          | (irrelevantAnswer, correctAnswer, incorrectAnswer) | (irrelevantAnswer, incorrectAnswer, correctAnswer) ]
```

or, using the interleave operator:

```
question[correctAnswer & incorrectAnswer & irrelevantAnswer]
```

- The interleave operator is **exponentially** more succinct
- We would like efficient static analyzers that directly operate on the succinct form! (i.e. not pay the price of the blow-up)

# The Counting Operator: Example

Querying all the articles with 4 or more authors

- Navigational XPath expression:

```
article[author/following-sibling::author/following-sibling::author/following-sibling::author]
```

or, using the counting operator in XPath:

```
article[count(author)>=4]
```

- The counting operator is **exponentially** more succinct
- Again, we would like efficient static analyzers that directly operate on the succinct form! (i.e. not pay the price of the blow-up)

# Contribution

A tree logic equipped with a notion of counting:

$$\varphi := \top \mid \perp \mid p \mid \neg p \mid x \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \langle m \rangle \varphi \mid \neg \langle m \rangle \top \mid \mu x. \varphi \mid \varphi^{\langle k \rangle} \mid \varphi^{[k]}$$

where  $m \in \{\downarrow, \rightarrow, \uparrow, \leftarrow\}$

## New Constructs

- $\varphi^{\langle k \rangle}$  : at least  $k+1$  children nodes satisfy  $\varphi$
- $\varphi^{[k]}$  : at most  $k$  children nodes satisfy  $\varphi$

## Example

$a \wedge b^{\langle 3 \rangle}$  denotes the set of nodes labeled “ $a$ ” with at least 4 children labeled “ $b$ ”

## Interleaving and Counting are Closely Related

- The logic supports a (restricted) form of interleaving occurring in schemas (where interleaved operands must be disjunction-free)
- Such interleaving is translated in terms of counting, e.g.:

$$q[a_1 \& a_2 \& a_3] = q \wedge (a_1)^{=1} \wedge (a_2)^{=1} \wedge (a_3)^{=1} \wedge \top^{=3} \wedge \neg \langle \rightarrow \rangle \top$$

- The general translation relies on **nominals** to mark occurrences of the same label that must be distinguished (as in, e.g.,  $q[a_1 \& a_2 \& a_1]$ )
- A **nominal**  $\text{nom}$  (in a tree) can be expressed with a constant-size formula:

$$\begin{aligned} \text{nom} &= \text{nom} \wedge \neg \text{descendant}(\text{nom}) \\ \wedge \neg \text{descendant-or-self}(\text{following-sibling}(\text{ancestor-or-self}(\text{nom}))) \end{aligned}$$

# Results on Interleaving and Counting

- Satisfiability for the tree logic is decidable and EXPTIME-hard
  - Bottom-up proof construction algorithm à la Fisher-Ladner inspired from the constructive algorithm of [Geneves-PLDI'07]
- deriving an implementation is straightforward
- A logic with similar features is undecidable over graphs [Bonatti-AI'04]

The XML Reasoning Project:  
<http://wam.inrialpes.fr/websolver>