Efficient Static Analysis of XML Paths and Types

Pierre Genevès – EPFL, Switzerland

Joint work with Nabil Layaïda and Alan Schmitt – INRIA, France

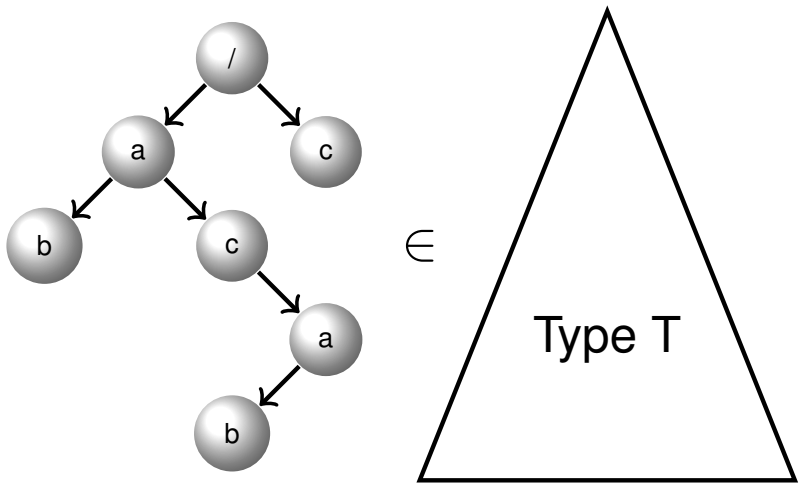
PLDI'07, San Diego, June 2007

Introduction

- More and more XML data
- **Objective:** ensuring safety and efficiency of programs that manipulate XML
- Two ways for processing XML:
  1. General purpose languages extended with librairies
  2. DSLs: e.g. **XSLT, XQuery** (W3C standards) that rely on **XPath**
- In both cases: static analysis of programs very hard (very complex to detect errors at compile-time)
- **This paper:** we solve important XML static analysis tasks by reduction to satisfiability of a new tree logic

Safety and Efficiency of Programs

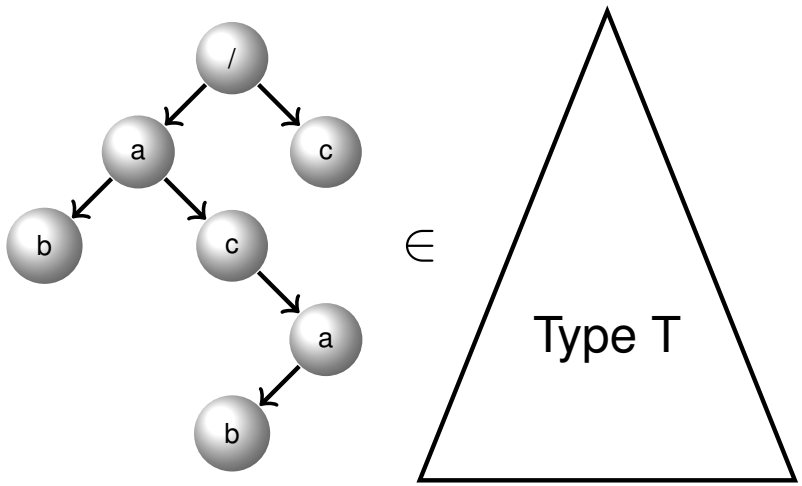
- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)

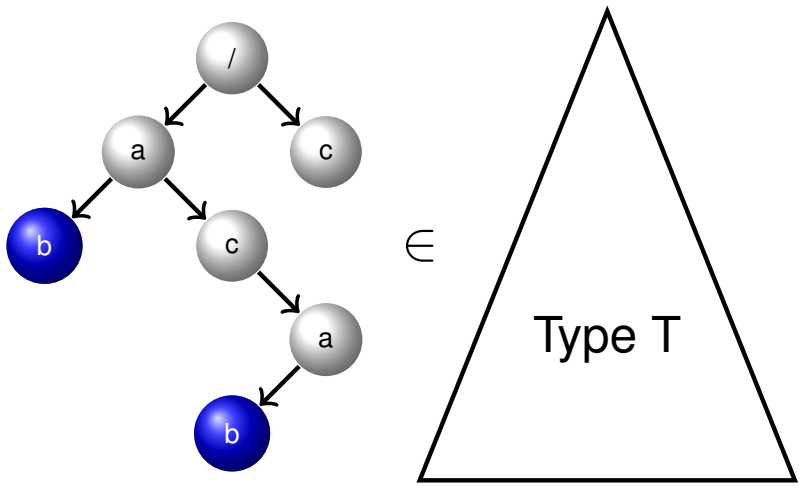


/descendant::b/parent::a/child::c

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)

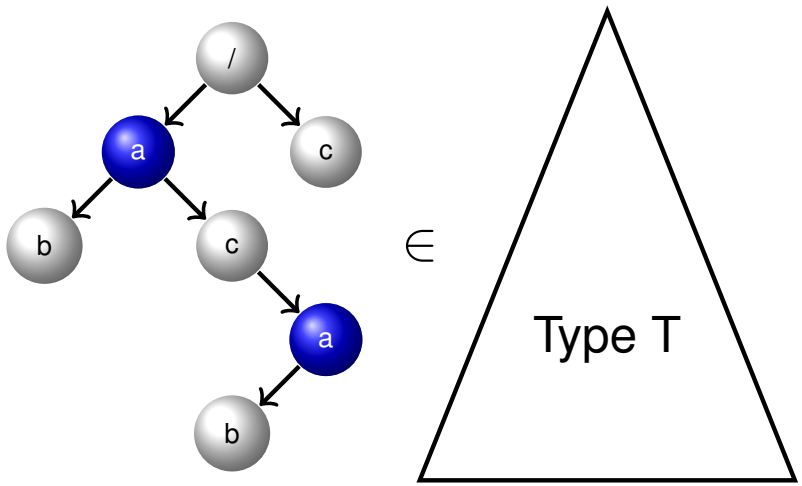


/descendant::b/parent::a/child::c

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)

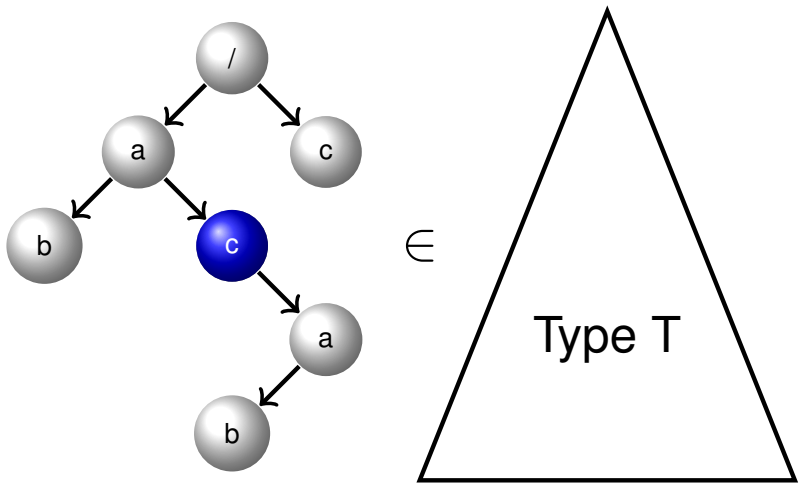


/descendant::b/parent::a/child::c

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)

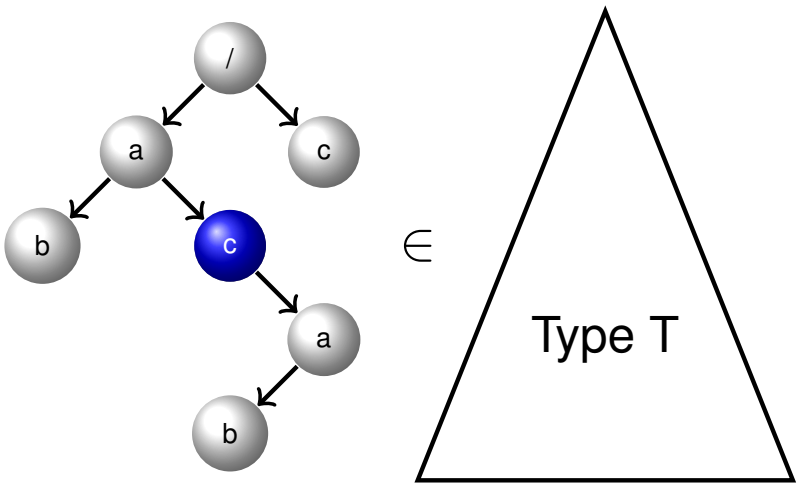


/descendant::b/parent::a/child::c

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q}$$

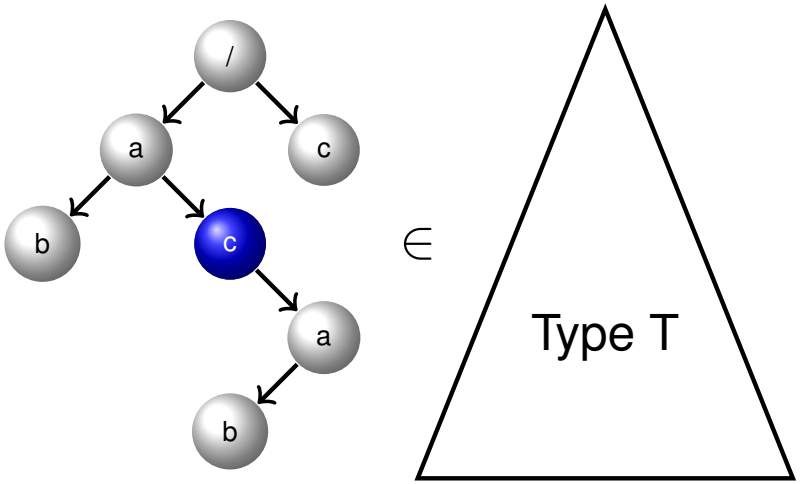
```
for x in (q) do  {
    ...
}
```

```
let n = q; ...
```

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \oplus T \stackrel{?}{=} \emptyset$$

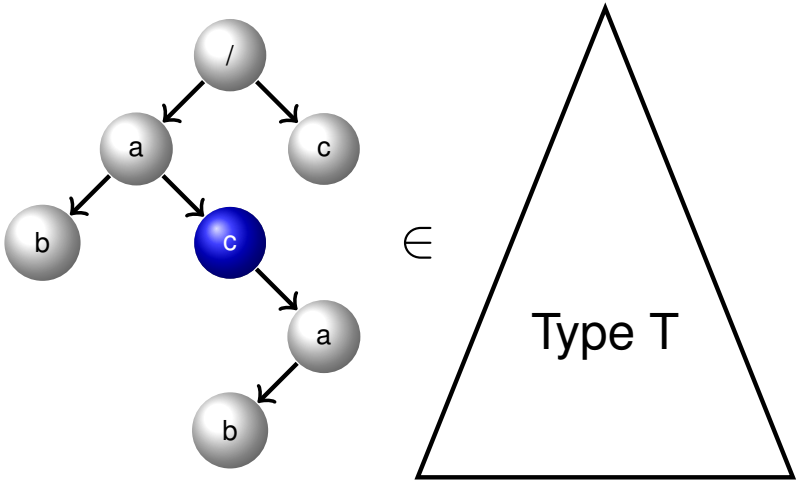
```
for x in (q) do  {
    ...
}
```

```
let n = q; ...
```

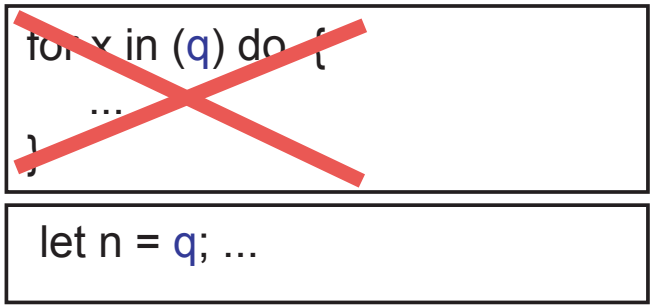
- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



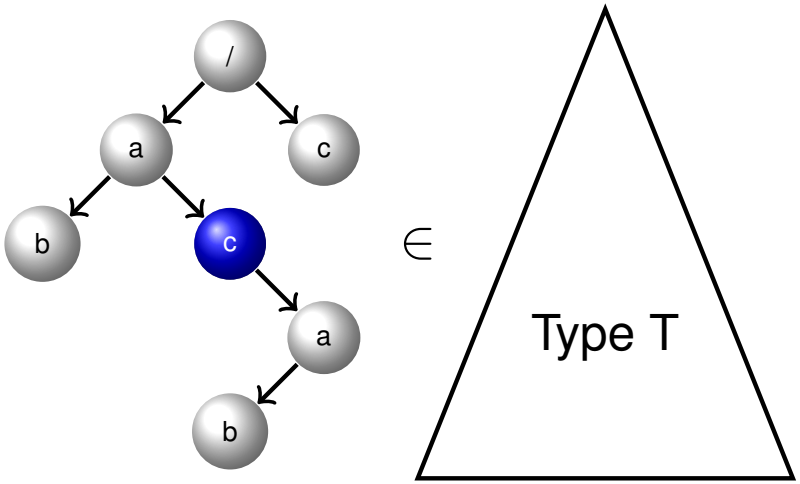
$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \oplus T \stackrel{?}{=} \emptyset$$



- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \quad \oplus T \stackrel{?}{\equiv} \underbrace{\texttt{/child::a/child::c}}_{q_{\text{optimised}}}$$

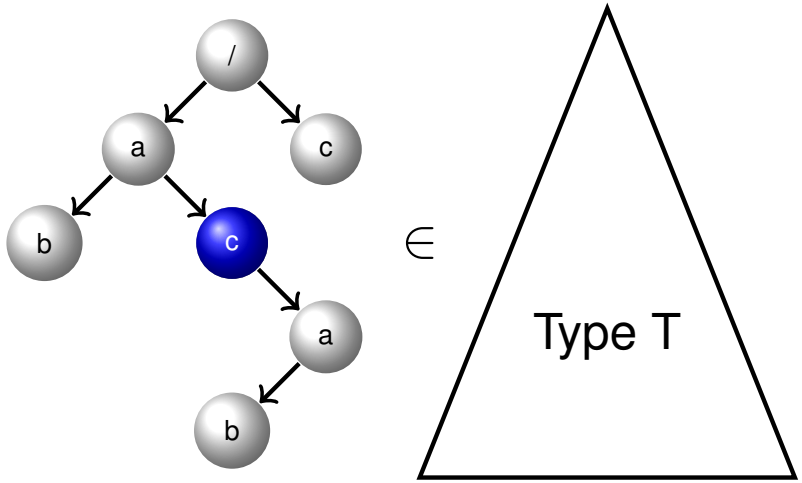
```
for x in (q) do  {
    ...
}
```

```
let n = q; ...
```

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \quad \oplus T \stackrel{?}{\equiv} \underbrace{\texttt{/child::a/child::c}}_{q_{\text{optimised}}}$$

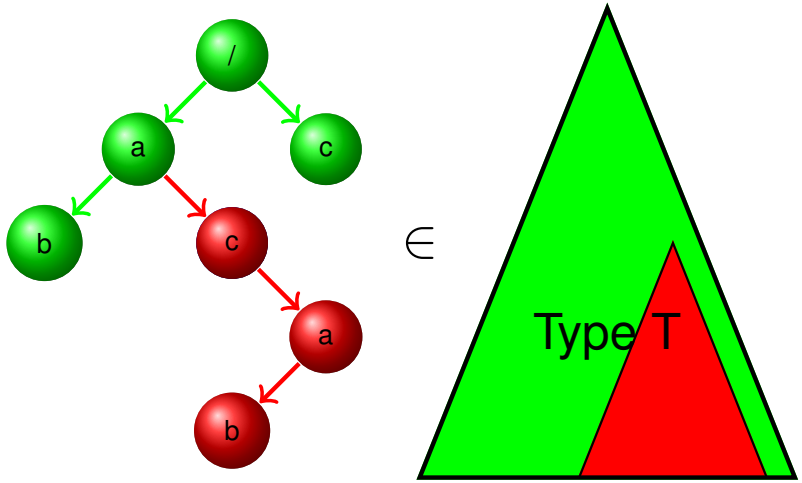
```
for x in (q) do  {
    ...
}
```
$q_{\text{optimised}}$

```
let n = q; ...
```

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \quad \oplus T \stackrel{?}{\equiv} \underbrace{\texttt{/child::a/child::c}}_{q_{\text{optimised}}}$$

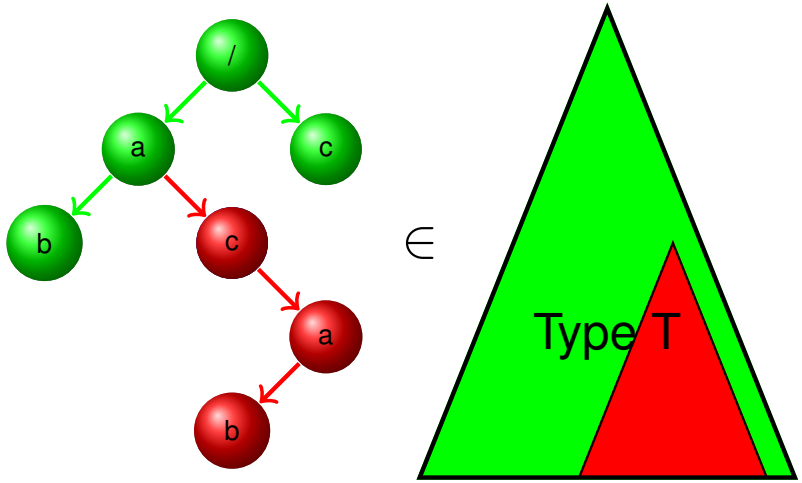
```
for x in (q) do  {
      ...
}
```
$q_{\text{optimised}}$

```
let n = q; ...
```

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \quad \oplus T \stackrel{?}{\equiv} \underbrace{\texttt{/child::a/child::c}}_{q_{\text{optimised}}}$$

$$q \cap q_{\text{forbidden}} \stackrel{?}{\neq} \emptyset$$

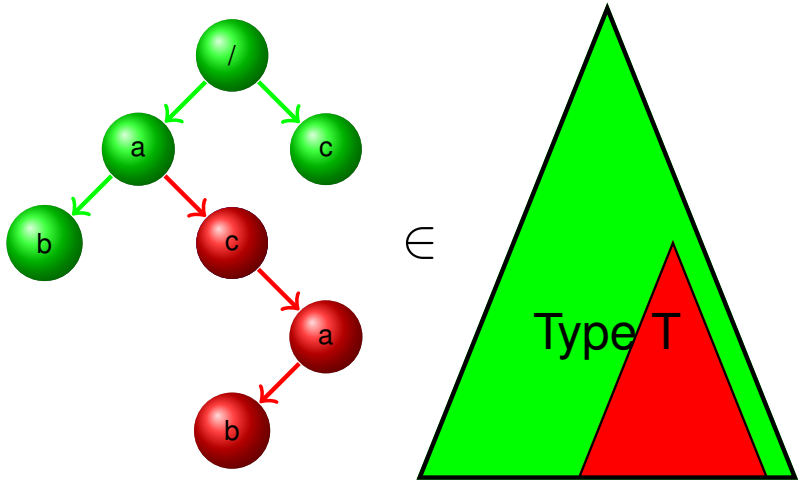
```
for x in (q) do  {
    ...
}                    qoptimised
```

```
let n = q; ...
```

- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

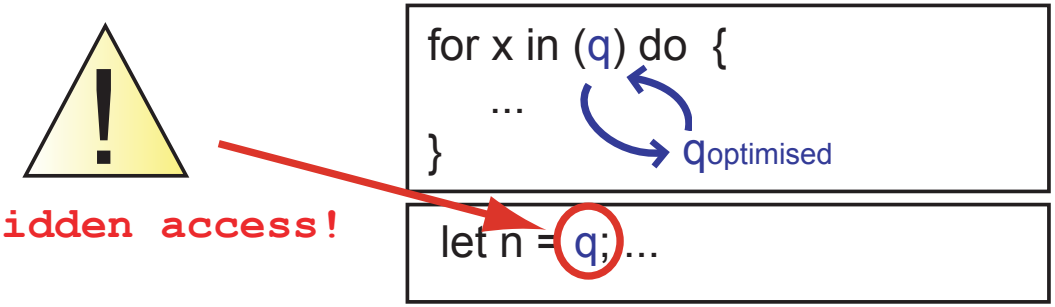
Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \quad \oplus T \stackrel{?}{\equiv} \underbrace{\texttt{/child::a/child::c}}_{q_{\text{optimised}}}$$

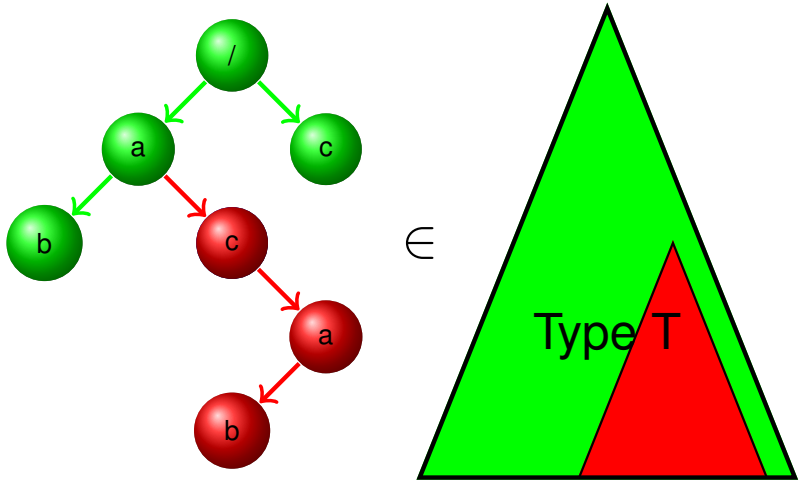
$$q \cap q_{\text{forbidden}} \stackrel{?}{\neq} \emptyset$$



- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

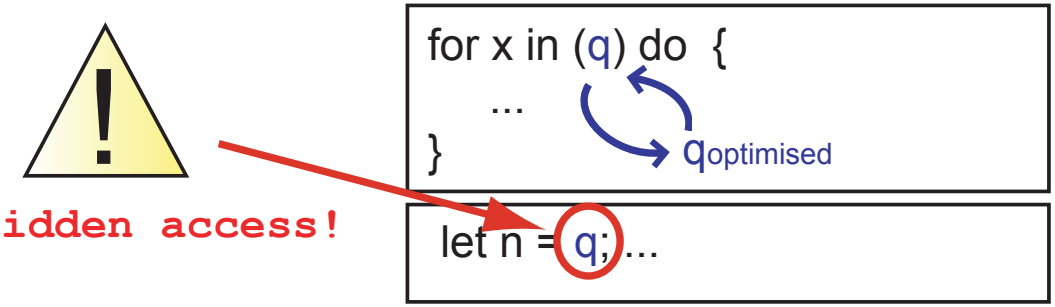
Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \quad \oplus T \stackrel{?}{\equiv} \underbrace{\texttt{/child::a/child::c}}_{q_{\text{optimised}}}$$

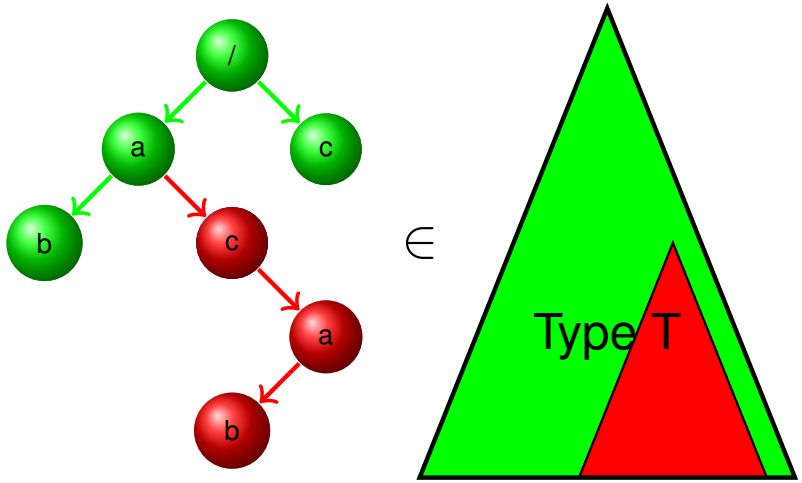
$$q \cap q_{\text{forbidden}} \stackrel{?}{\neq} \emptyset$$



- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

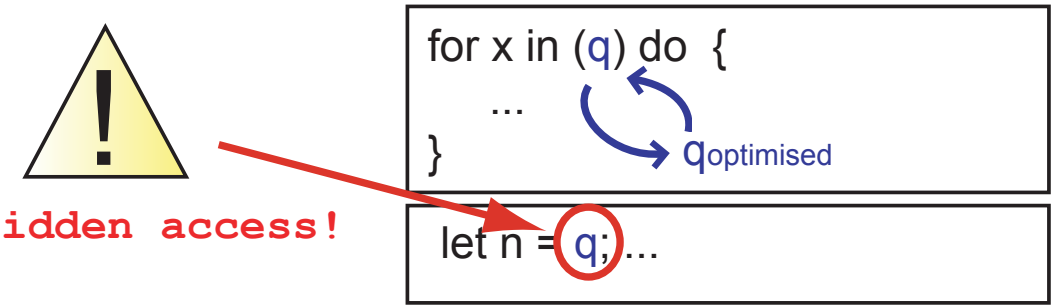
Safety and Efficiency of Programs

- Programs that manipulate XML trees
- Analysis:
  - tree types (XML Schemas, DTDs)
  - queries (XPath)



$$\underbrace{\texttt{/descendant::b/parent::a/child::c}}_{q} \quad \oplus T \stackrel{?}{\equiv} \underbrace{\texttt{/child::a/child::c}}_{q_{\text{optimised}}}$$

$$q \cap q_{\text{forbidden}} \stackrel{?}{\neq} \emptyset$$



- **Before:** complexity too high, implementations out of scope...
- **This paper:** optimal complexity + efficient implementation

XPath Static Analysis Tasks

Basic Tasks

1. XPath typing
2. XPath query comparisons
   - query containment, emptiness, overlap, equivalence

Main Applications

- Static analysis of host languages: error detection, optimization (static type-checkers, optimizing compilers)
- Checking integrity constraints in XML databases

Challenges

- Query comparisons and typing are undecidable for the complete XPath language

Open Questions

- What are the largest XPath fragments with decidable static analysis?
- Which fragments can be effectively decided in a compiler?
- Is there a generic algorithm able to solve all related XPath decision problems?

Difficulties

- Considered XPath operators and their combination (e.g., multidirectional navigation, recursion)
- Checking properties on a possibly infinite set of XML documents
- Very high computational complexity

Challenges

- Query comparisons and typing are undecidable for the complete XPath language

Open Questions

- What are the largest XPath fragments with decidable static analysis?
- Which fragments can be effectively decided in a compiler?
- Is there a generic algorithm able to solve all related XPath decision problems?

Difficulties

- Considered XPath operators and their combination (e.g., multidirectional navigation, recursion)
- Checking properties on a possibly infinite set of XML documents
- Very high computational complexity

Challenges

- Query comparisons and typing are undecidable for the complete XPath language

Open Questions

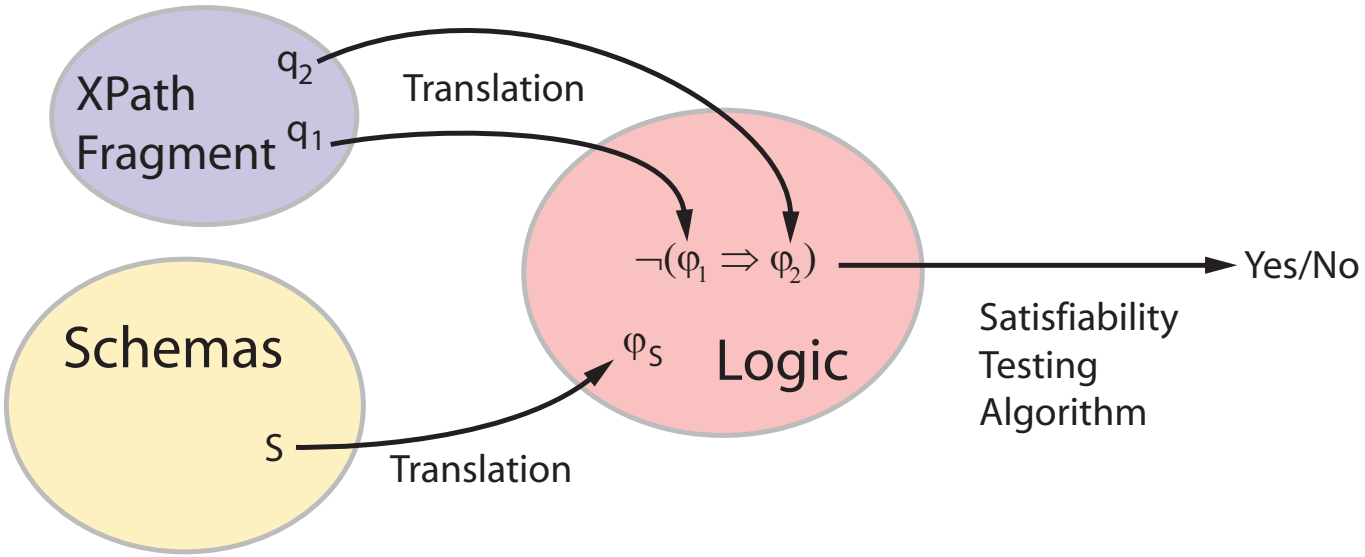
- What are the largest XPath fragments with decidable static analysis?
- Which fragments can be effectively decided in a compiler?
- Is there a generic algorithm able to solve all related XPath decision problems?

Difficulties

- Considered XPath operators and their combination (e.g., multidirectional navigation, recursion)
- Checking properties on a possibly infinite set of XML documents
- Very high computational complexity

The Logical Approach: Overview

- Find an appropriate logic for reasoning on XML trees
- Formulate the problem into the logic and test satisfiability



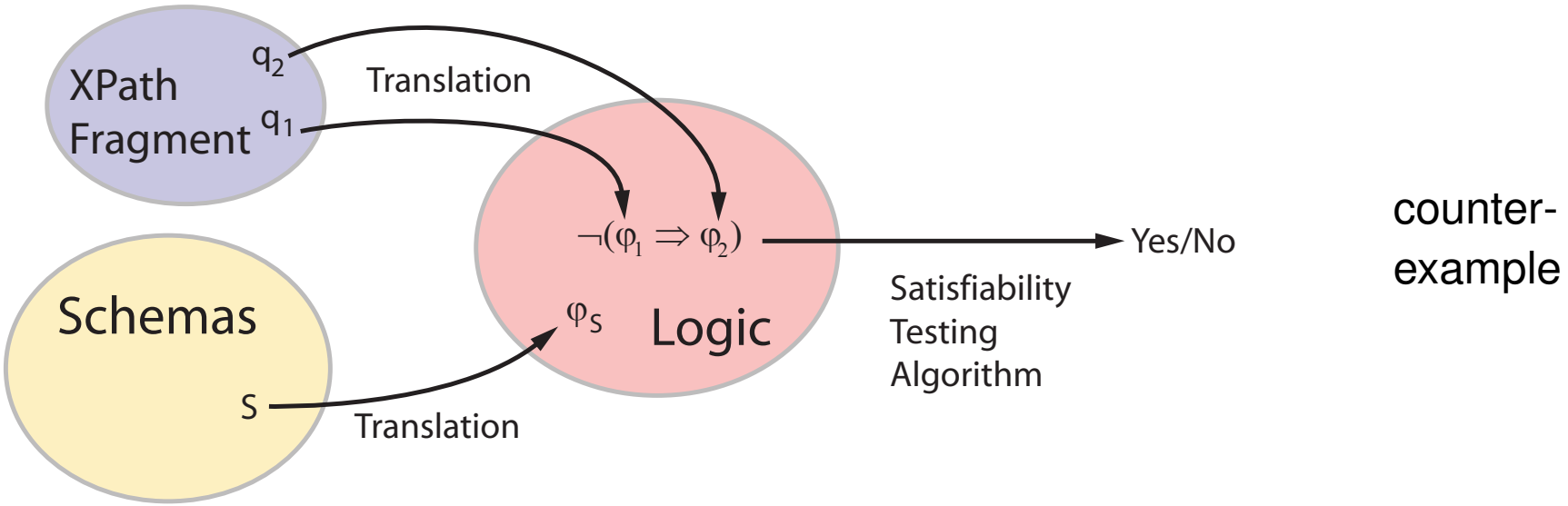
counter-example

Critical Aspects

1. The logic must be expressive enough
2. The algorithm must be effective in practice for XML translations

The Logical Approach: Overview

- Find an appropriate logic for reasoning on XML trees
- Formulate the problem into the logic and test satisfiability

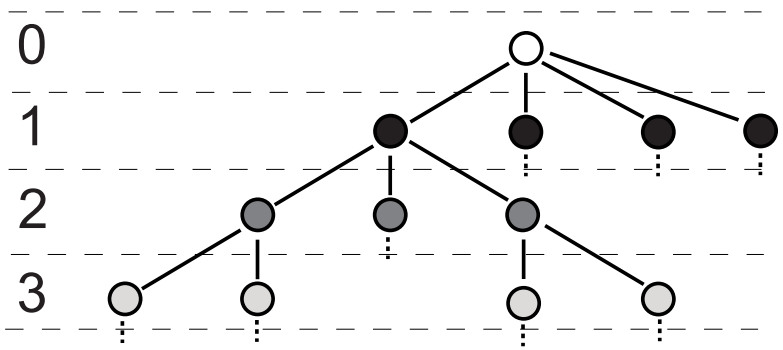


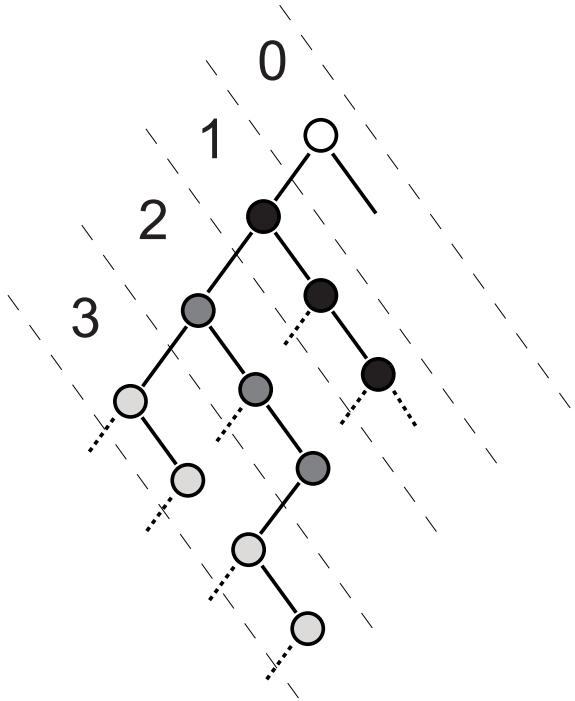
Critical Aspects

1. The logic must be expressive enough
2. The algorithm must be effective in practice for XML translations

Models for XML Documents

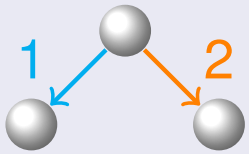
- Finite ordered binary trees, one label per node
- Bijective encoding of unranked trees as binary trees:





Formulas of the $\mathcal{L}_\mu$ Logic

- Programs $\alpha \in \{1, 2, \overline{1}, \overline{2}\}$ for navigating binary trees ($\overline{\overline{\alpha}} = \alpha$)



$\mathcal{L}_\mu \ni \varphi, \psi$	$::=$	formula
	$\top$	true
$\mid$	$\sigma \quad \mid \quad \neg\sigma$	atomic prop (negated)
$\mid$	$\textcircled{S} \quad \mid \quad \neg\textcircled{S}$	starting context (negated)
$\mid$	$\varphi \vee \psi$	disjunction
$\mid$	$\varphi \wedge \psi$	conjunction
$\mid$	$\langle\alpha\rangle\,\varphi \quad \mid \quad \neg\langle\alpha\rangle\,\top$	existential (negated)
$\mid$	X	variable
$\mid$	$\mu X.\varphi$	unary fixpoint
$\mid$	$\mu\overline{X_i.\varphi_i}$ in ψ	n-ary fixpoint

- Closed formulas

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*

in $\mathcal{L}_\mu$:

- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*

in $\mathcal{L}_\mu$: *a*

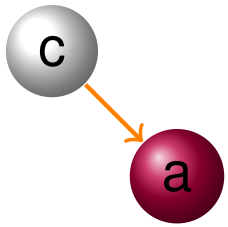
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*

in $\mathcal{L}_\mu$: $\langle 2 \rangle a$



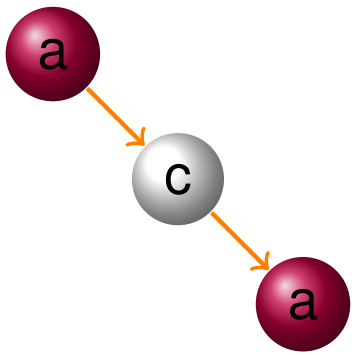
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*

in $\mathcal{L}_\mu$: $\langle 2 \rangle a$



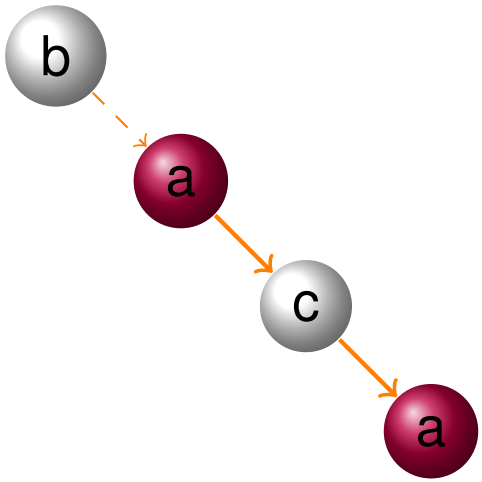
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*

in $\mathcal{L}_\mu$: $\langle 2 \rangle a$



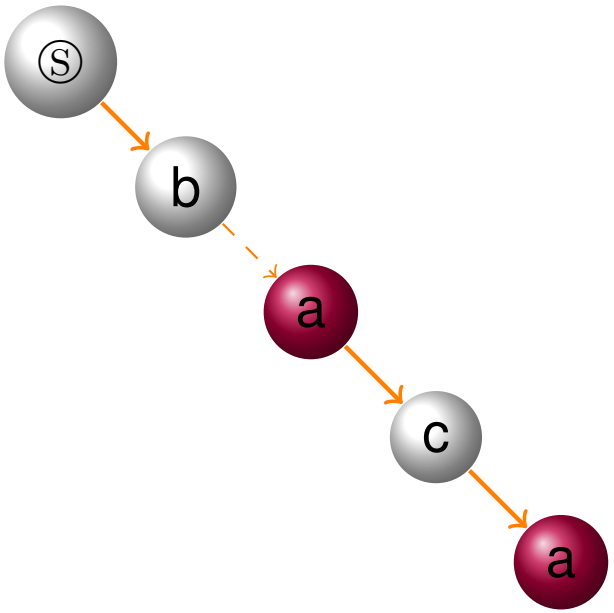
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*

in $\mathcal{L}_\mu$: $a \wedge (\mu Z.\, \langle \overline{2} \rangle \, \mathbb{S} \vee \langle \overline{2} \rangle \, Z)$



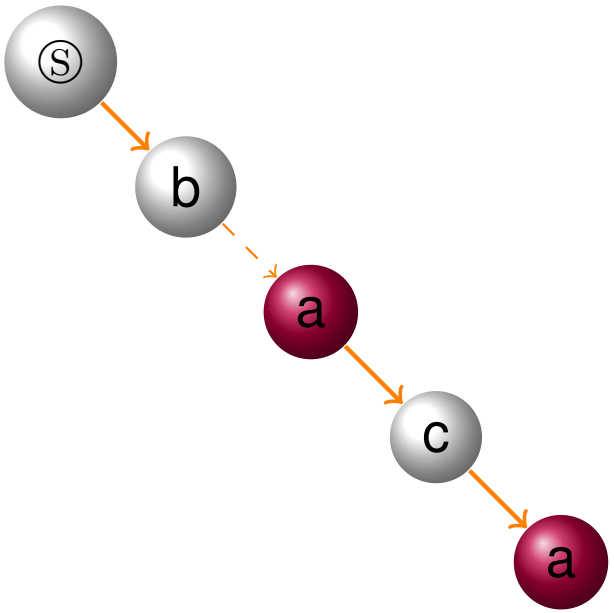
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*/preceding-sibling::*b*

in $\mathcal{L}_\mu$: $a \wedge (\mu Z.\, \langle \overline{2} \rangle \, \mathbb{S} \vee \langle \overline{2} \rangle \, Z)$



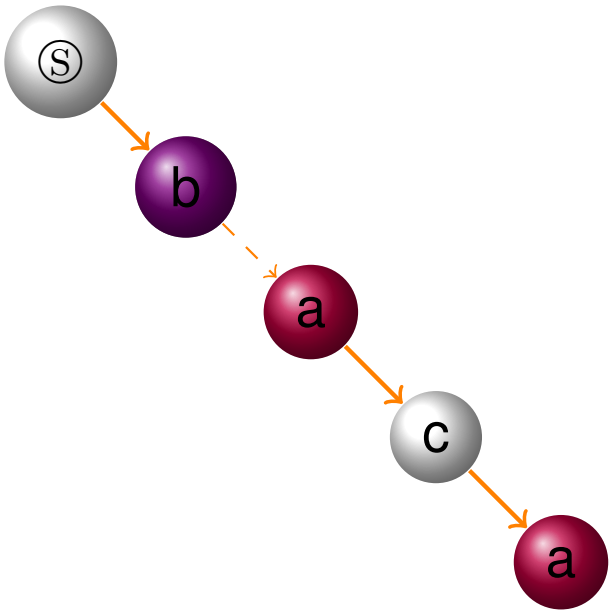
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*/preceding-sibling::*b*

in $\mathcal{L}_\mu$: $b \wedge [\mu Y.\, \langle 2 \rangle\, (a \wedge (\mu Z.\, \langle \overline{2} \rangle\, \textcircled{s} \vee \langle \overline{2} \rangle\, Z))\, \vee \langle 2 \rangle\, Y]$



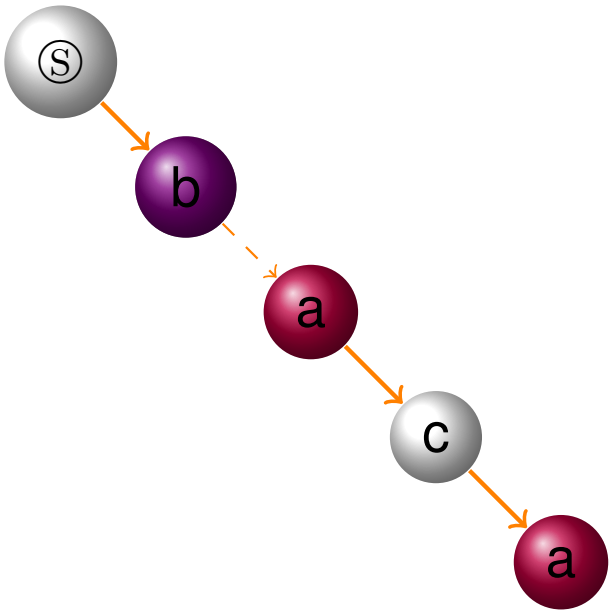
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*/preceding-sibling::*b*

in $\mathcal{L}_\mu$: $b \wedge [\mu Y.\, \langle 2 \rangle\, (a \wedge (\mu Z.\, \langle \overline{2} \rangle\, \mathbb{S} \vee \langle \overline{2} \rangle\, Z)) \vee \langle 2 \rangle\, Y]$



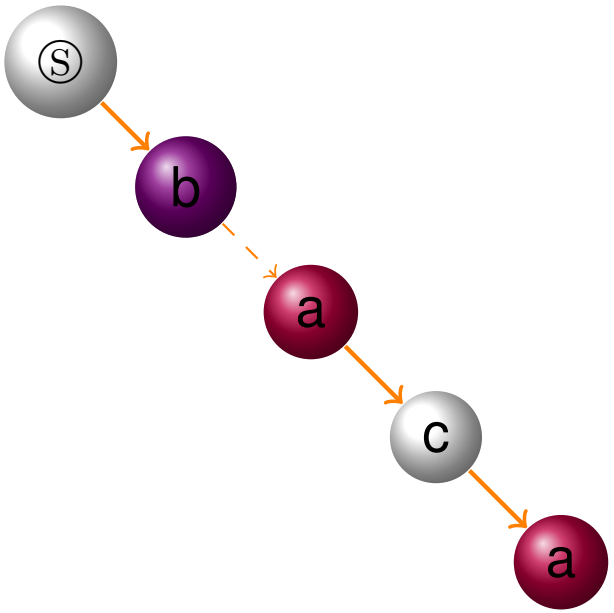
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*/preceding-sibling::*b*

in $\mathcal{L}_\mu$: $b \wedge [\mu Y.\, \langle 2 \rangle\, (a \wedge (\mu Z.\, \langle \overline{2} \rangle\, \mathbb{S} \vee \langle \overline{2} \rangle\, Z)) \vee \langle 2 \rangle\, Y]$



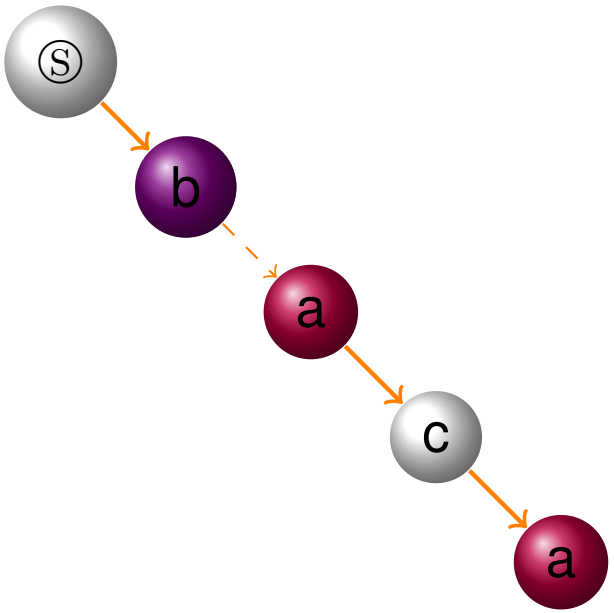
- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Semantics of $\mathcal{L}_\mu$

- The set of models of a formula φ is the set of finite binary trees for which φ is satisfied on some node

Translating following-sibling::*a*/preceding-sibling::*b*

in $\mathcal{L}_\mu$: $b \wedge [\mu Y.\, \langle 2 \rangle\, (a \wedge (\mu Z.\, \langle \overline{2} \rangle\, \mathbb{S} \vee \langle \overline{2} \rangle\, Z)) \vee \langle 2 \rangle\, Y]$



- $\mu Z.\varphi$: finite recursion
- $\{\overline{1}, \overline{2}\}$ required for forward axes!
- $\{1, 2\}$ required for reverse axes!
- Converse programs are crucial
- Almost full XPath can be translated (only variable counting constraints and data value comparisons left)
- Schemas can also be captured!

Satisfiability-Testing Algorithm: Principles

Search for a Tree that Satisfies ψ

- ψ truth status can be determined from a few of its subformulas
- A node is a ψ-type (conjunction of formulas)

Bottom-up Construction of a Tree of ψ-types

- A set T of ψ-types is repeatedly updated (least fixpoint computation)
  - Initially: $\emptyset$
  - Step 1 : all possible leaves are added
  - Step i : all possible parent nodes of current nodes are added

Termination

- If ψ is present in some node, then ψ is satisfiable
- Otherwise, the algorithm terminates when no more node can be added

Correctness & Complexity

Theorem

The satisfiability problem for a formula $\psi \in \mathcal{L}_\mu$ is decidable in time $2^{O(n)}$ where n is the size of ψ.

Experimental Results

The First Implementation

- Able to handle such a large XPath fragment
- Able to handle schemas (regular tree types)

What Can Now Be Done

Time (s)	Solved Problems
< 0.5	Comparisons of XPath queries (XPathmark) without tree types
< 1	Medium tree types involved ($\approx$ 30 symbols, $\approx$ 20 variables) Example: W3C SMIL
< 3	Large tree types involved ($\approx$ 100 symbols, $\approx$ 400 variables) Example: W3C XHTML

Summary and Perspectives

A New Tree Logic

- Best balance known between expressiveness/complexity
- Translation of main XML concepts: linear
- Implementation already fairly efficient for static analysis

Future Work

- Extensions of the logic
  - Decidable data-value comparisons
  - Decidable counting constraints
- Type inference for XSLT/XQuery without output type annotations
- More applications in program analysis?
  - $\mathcal{L}_\mu$ is as expressive as MSO, and the solver is orders of magnitude faster than MONA...

Thank you!

pierre.geneves@epfl.ch