Course: Advanced Static Analysis for XML

Pierre Genevès
CNRS

University of Grenoble Alpes, 2017–2018

Some Objectives

- Understand the expressive power and complexity of XML (NoSQL) languages
- Identify interesting fragments/extensions
- Answer new programming needs due to the evolution of World Wide Web
- Contribute to the standardization effort... and to the next generation Web
- Research crossroad between DB, PL, Web...

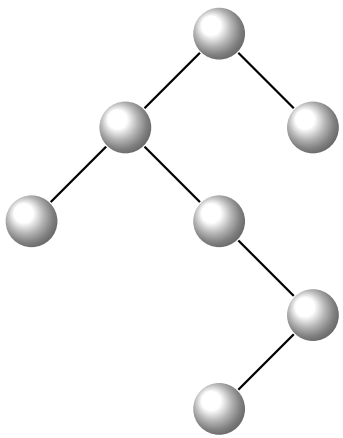
Some Challenges

- Ensure that programs safely and efficiently manipulate XML data

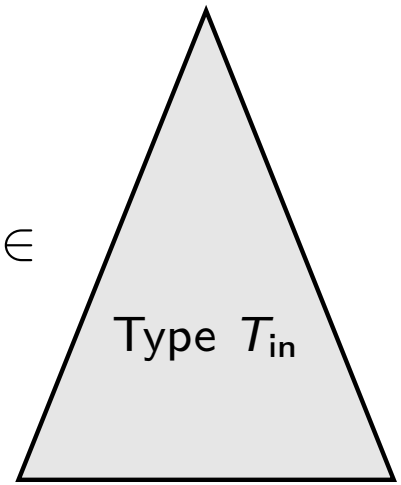
$\rightarrow$ type systems, static analyzers, optimizing compilers

- Introduce XML/XPath as first-class citizens in programming languages
- Ultimately: strongly typed XML processing (sound, precise, limited annotations)

Examples



$\in$

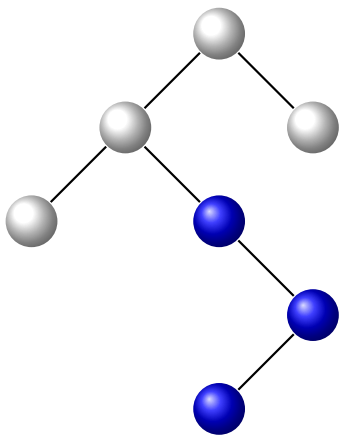



```
Transform(t, t')
...
Validate(t)
...
For x in (q) do {
...
}
let y=x;
Write(y, t')
...
Validate(t')
```

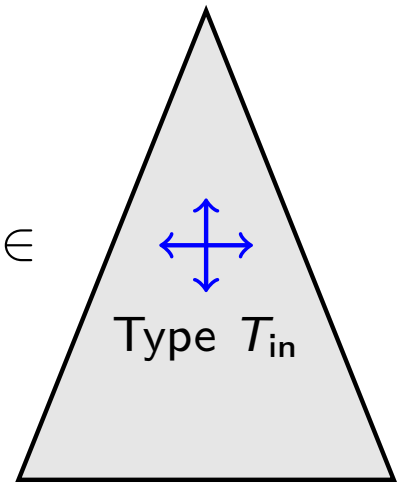
$\forall t \in T_{\text{in}}$, does $P(t) \in T_{\text{out}}$?

- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Examples



$\in$

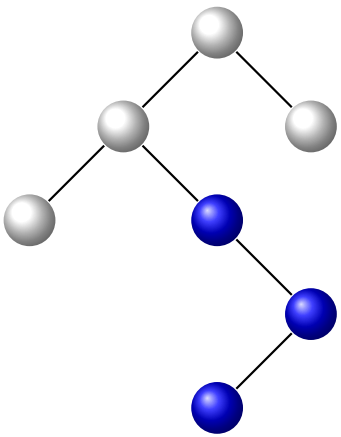


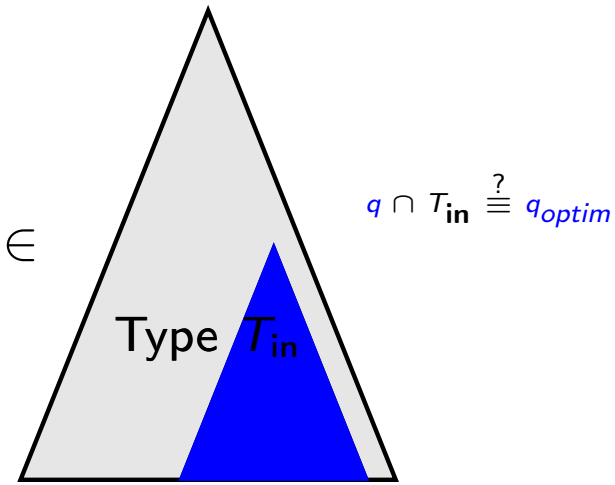
$q \cap T_{\text{in}} \overset{?}{\neq} \emptyset$

```
Transform(t, t')
...
Validate(t)
...
For x in (q) do {
...
}
let y=x;
Write(y, t')
...
Validate(t')
```

- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Examples



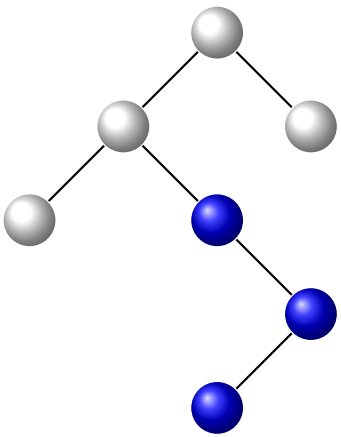



```
Transform(t, t')
...
Validate(t)
...
For x in (q) do {
...
}
let y=x;
Write(y, t')
...
Validate(t')
```

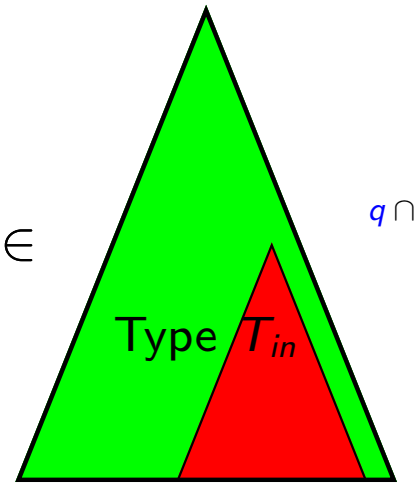
q_{optim}

- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Examples



$\in$

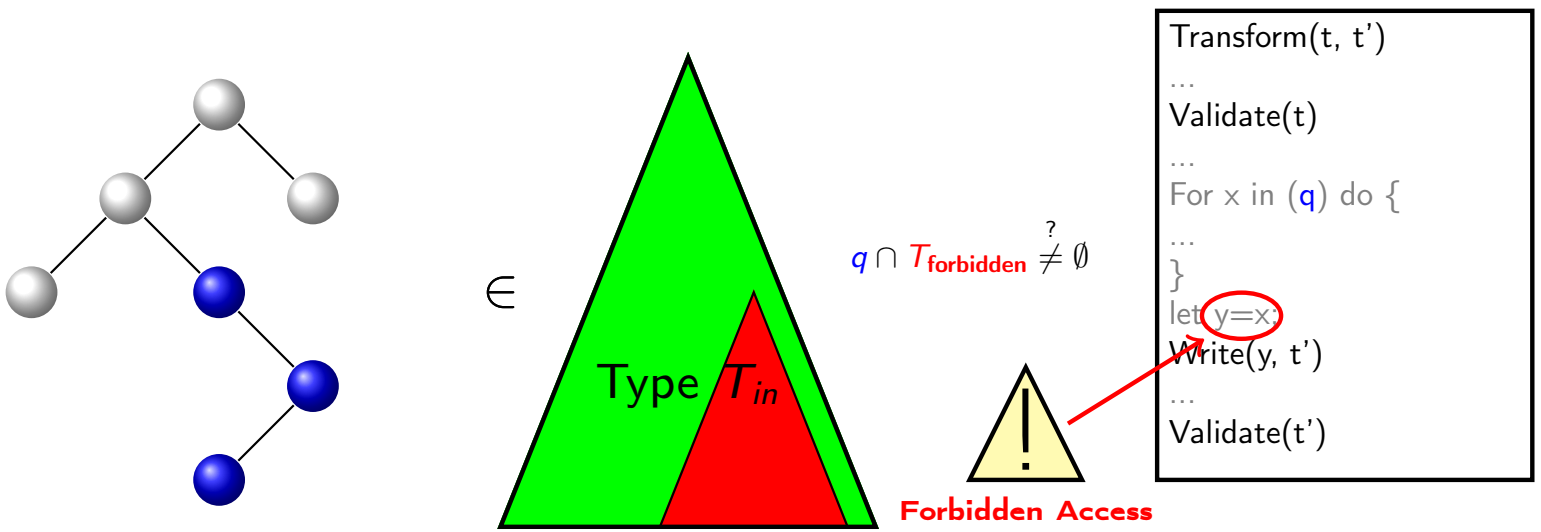


$q \cap T_{\text{forbidden}} \overset{?}{\neq} \emptyset$

```
Transform(t, t')
...
Validate(t)
...
For x in (q) do {
...
}
let y=x;
Write(y, t')
...
Validate(t')
```

- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Examples



- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Scientific Difficulties and Challenges

Key (and hard) problem

- Reasoning with XML types and (XPath) queries

Why is it computationally hard?

- $\mathcal{L}(T_1) \subseteq \mathcal{L}(T_2)$ is already in EXPTIME...
- Decidability envelope: XPath + (counting | comparisons) undecidable
- Infinite quantification: $\forall t, (t \in \mathcal{L}(T_{in}) \Rightarrow f(t) \in \mathcal{L}(T_{out}))$, algorithms?

- Prove properties for a set of XML documents (infinite quantification over trees)
- Need for appropriate (partly missing) theoretical tools...
- Foundations to be built!

Theoretical Models: Alternatives

Finite Tree Automata

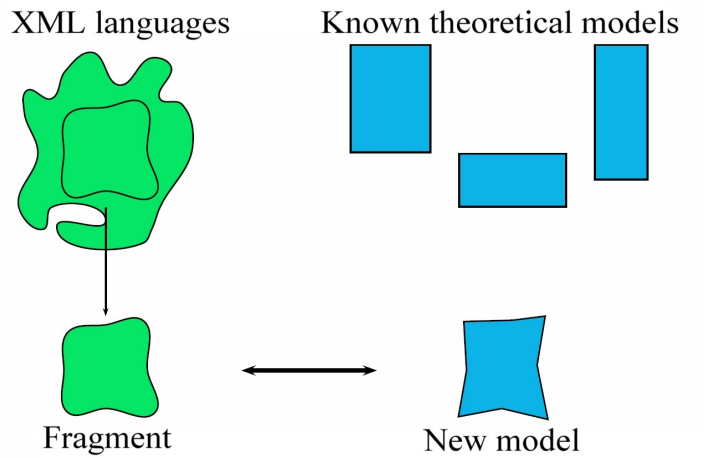
- ✓ capture regular tree types
- × how to capture $\mathcal{L}_{\text{XPath}}$?

First-Order logic (FO) over trees

- ✓ equivalent to $\mathcal{L}_{\text{XPath}}$
- × does not capture tree types

Monadic Second-Order logic (MSO/WS2S)

- ✓ FO with quant. over sets of nodes
- ✓ captures $\mathcal{L}_{\text{XPath}}$ and $\mathcal{L}_{\text{type}}$!
- × satisfiability is hyperexponential
- × blow-ups observable for XPath pbms



Our requirements:

1. Expressive enough to capture $\mathcal{L}_{\text{XPath}}$ and $\mathcal{L}_{\text{type}}$ but succinct
2. Best complexity
3. Nice algorithmic properties (not always worst case)

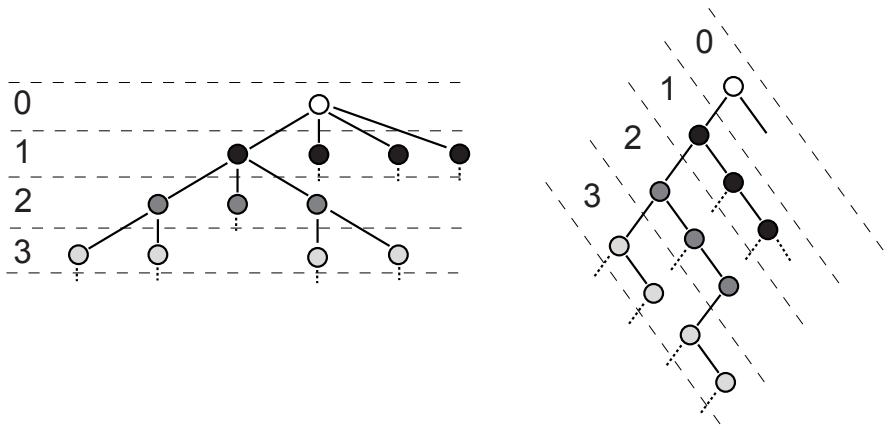
Outline

Some Results [Genevès et al., 2015]

1. An expressive yet efficient (modal) tree logic
2. An effective satisfiability-testing algorithm (using symbolic techniques)
3. Demo..

Data Model for the Logic

- XML trees are n-ary trees with one label per node
- There is a bijective encoding of unranked trees as binary trees

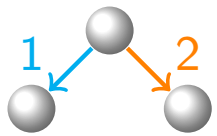


General Encoding

- Queries (binary relations on tree nodes)
- XML Types

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

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



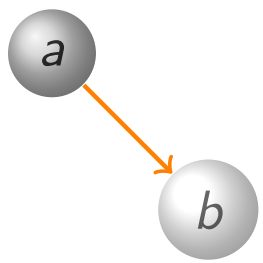
$\mathcal{L}_\mu \ni \varphi$	$::=$		formula
		$\top$	true
	$\mid$	$\sigma \mid \neg\sigma$	atomic prop (negated)
	$\mid$	$\varphi \vee \varphi \mid \varphi \wedge \varphi$	disjunction (conjunction)
	$\mid$	$\langle\alpha\rangle\,\varphi \mid \neg\langle\alpha\rangle\,\top$	existential (negated)
	$\mid$	$\mu X.\varphi$	unary fixpoint (finite recursion)
	$\mid$	$\mu\overline{X_i.\varphi_i}$ in φ	n-ary fixpoint

Sample Formula and Satisfying Tree

a

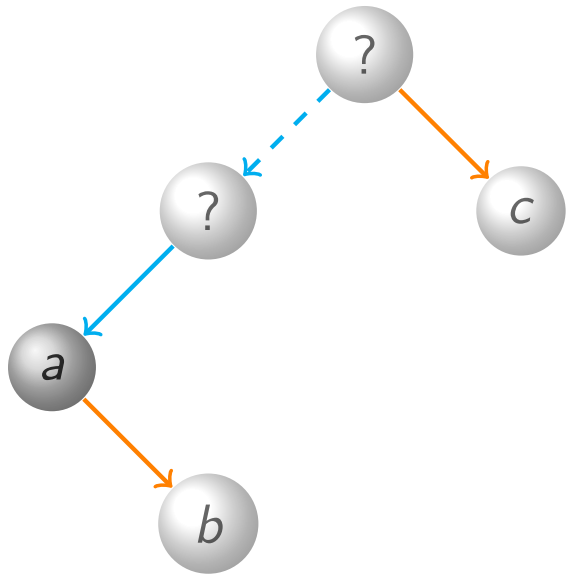
Sample Formula and Satisfying Tree

$a \wedge \langle 2 \rangle\, b$



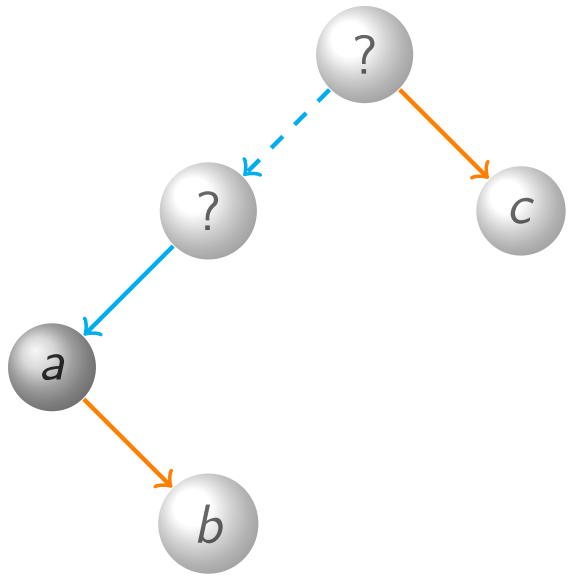
Sample Formula and Satisfying Tree

$$a \wedge \langle 2 \rangle\, b \wedge \mu X.\, \langle 2 \rangle\, c \vee \langle \overline{1} \rangle\, X$$



Sample Formula and Satisfying Tree

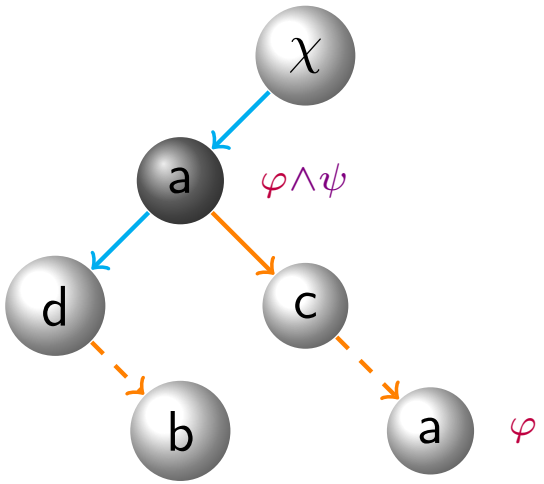
$$a \wedge \langle 2 \rangle\, b \wedge \mu X.\, \langle 2 \rangle\, c \vee \langle \overline{1} \rangle\, X$$



- Semantics: models of φ are finite trees for which φ holds at some node

✓ XPath and XML types can be translated into the logic, linearly

Translation of $\mathcal{L}_{\text{XPath}}$ into $\mathcal{L}_\mu$



- Formula holds at selected nodes
- $\mu Z.\varphi$: finite recursion
- Converse programs are crucial
- $t_{\text{xpath}}(e, \chi) : \mathcal{L}_{\text{XPath}} \times \mathcal{L}_\mu \to \mathcal{L}_\mu$
- χ is the latest navigation step
  - Initially, χ navigates to the root for absolute paths (exercise)

Translated query: child::a [child::b]

$$\underbrace{a \wedge (\mu Z.\, \langle \overline{1} \rangle \chi \vee \langle \overline{2} \rangle Z)}_{\varphi} \quad \wedge \quad \underbrace{\langle 1 \rangle \mu Y.b \vee \langle 2 \rangle Y}_{\psi}$$

Outline

1. An expressive yet efficient (modal) tree logic
2. An effective satisfiability-testing algorithm (using symbolic techniques)
3. Demo..

Deciding Satisfiability

Is a formula ψ satisfiable?

- Given ψ, determine whether there exists a tree that satisfies ψ
- Validity: test $\neg\psi$
- Different (more complex) than model-checking

Principles: Automatic Theorem Proving

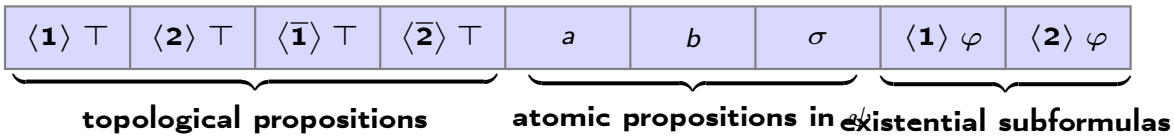
- Search for a proof tree
- Build the proof bottom up: if ψ holds then it is necessarily somewhere up

Search Space Optimization

Idea: Truth Status is Inductive

- The truth status of ψ can be expressed as a function of its subformulas
- For boolean connectives, it can be deduced (truth tables)
- Only base subformulas really matter: Lean(ψ)

Lean(ψ) :



A Tree Node: Truth Assignment of Lean(ψ) Formulas

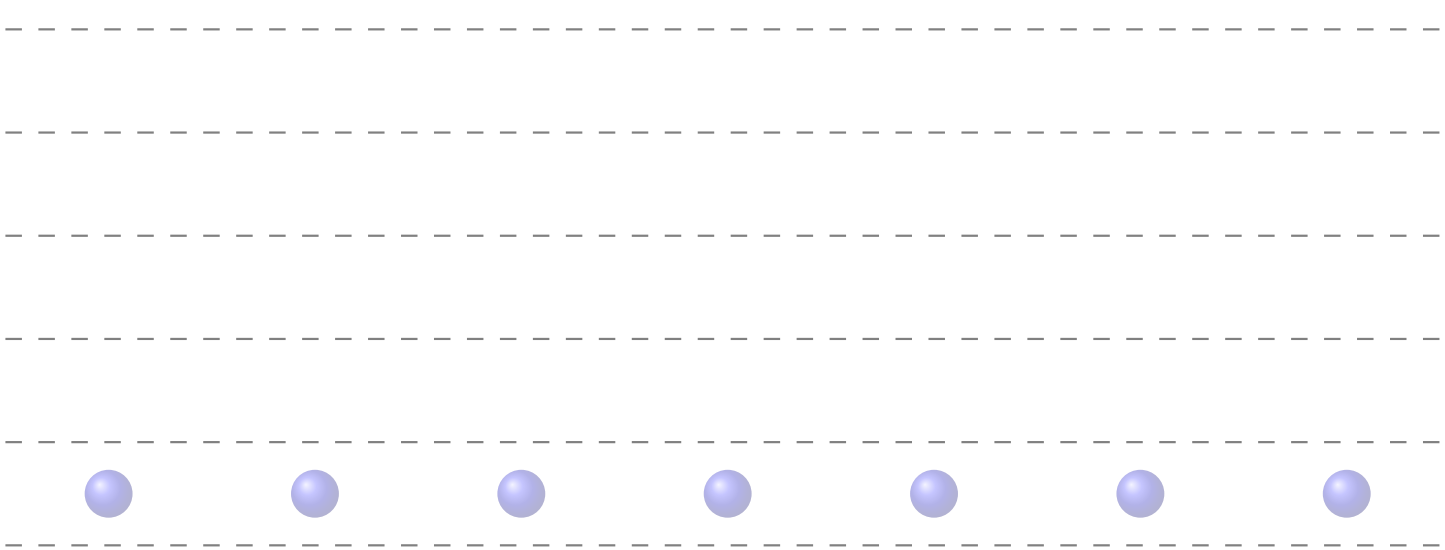
- With some additional constraints, e.g. $\neg \langle \overline{1} \rangle \top \vee \neg \langle \overline{2} \rangle \top$

Satisfiability-Testing Algorithm: Principles

Bottom-up construction of proof tree

- A set of nodes is repeatedly updated (fixpoint computation)

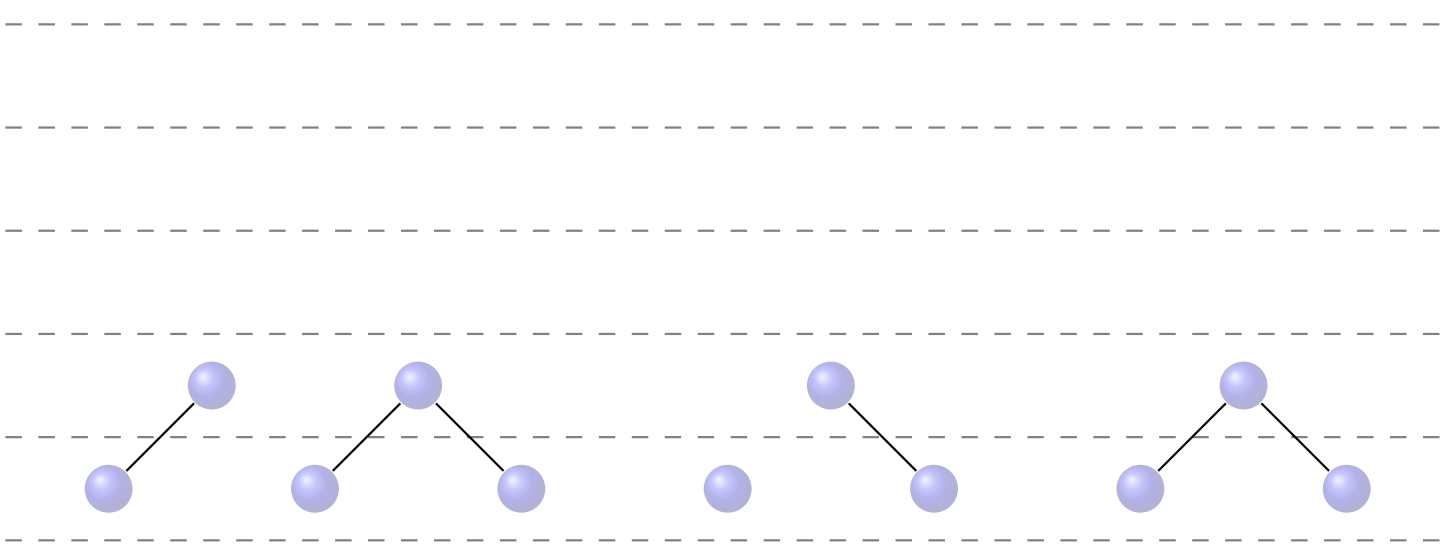
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

- Step 1: all possible leaves are added

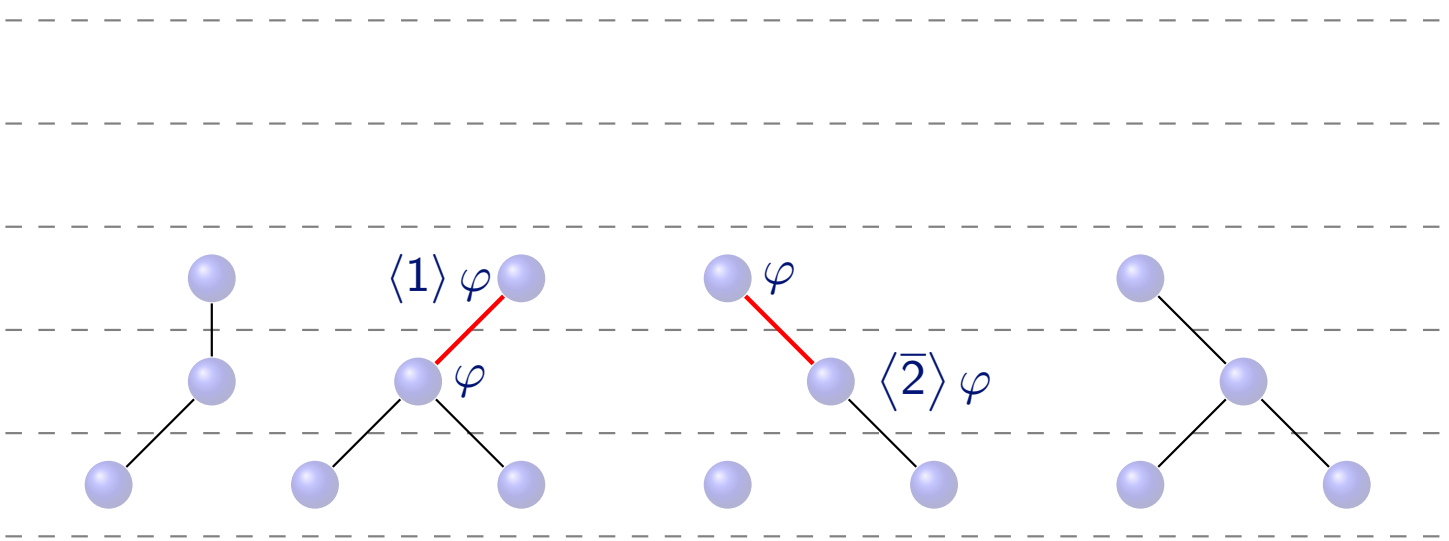
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

- Step $i > 1$: all possible parents of previous nodes are added

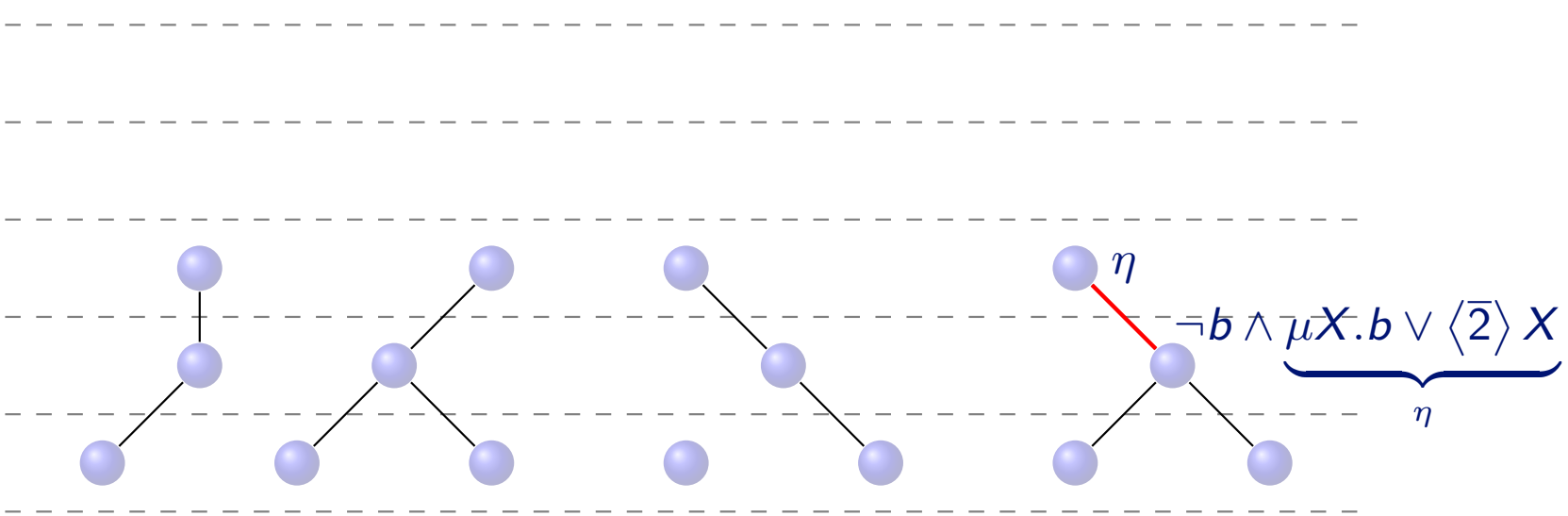
Satisfiability-Testing Algorithm: Principles



Compatibility relation between nodes

- Nodes from previous step are proof support:
  $\langle\alpha\rangle\,\varphi$ is added if φ holds in some node added at previous step

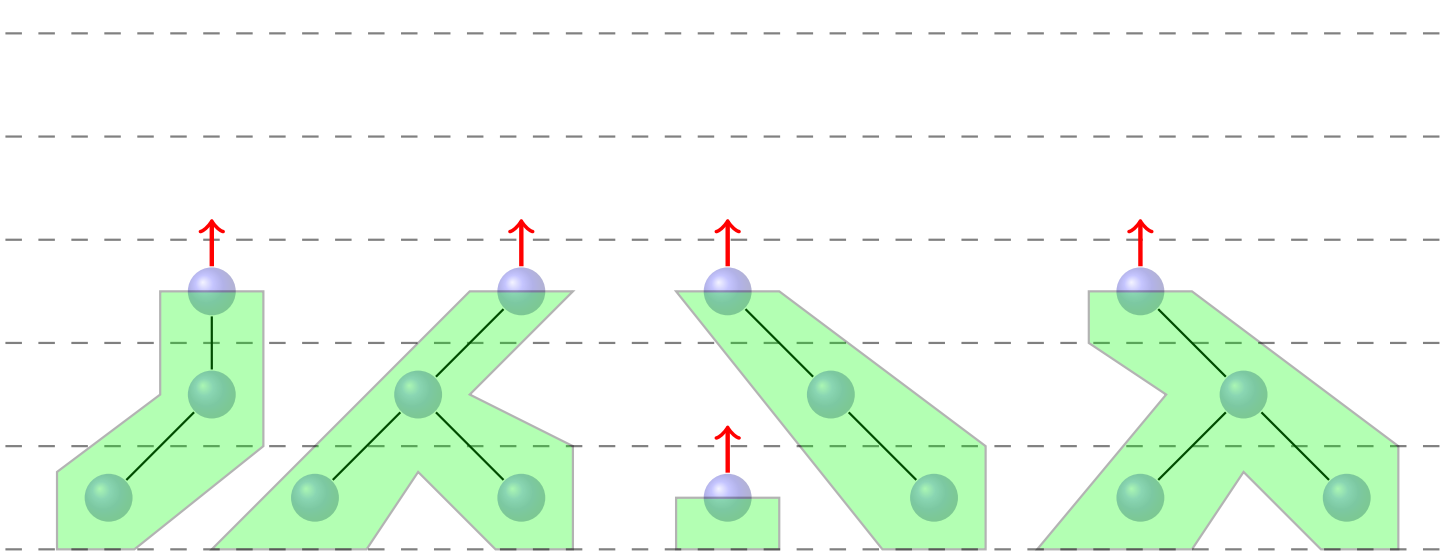
Satisfiability-Testing Algorithm: Principles



Compatibility relation between nodes

- Nodes from previous step are proof support:
  $\langle \alpha \rangle \varphi$ is added if φ holds in some node added at previous step

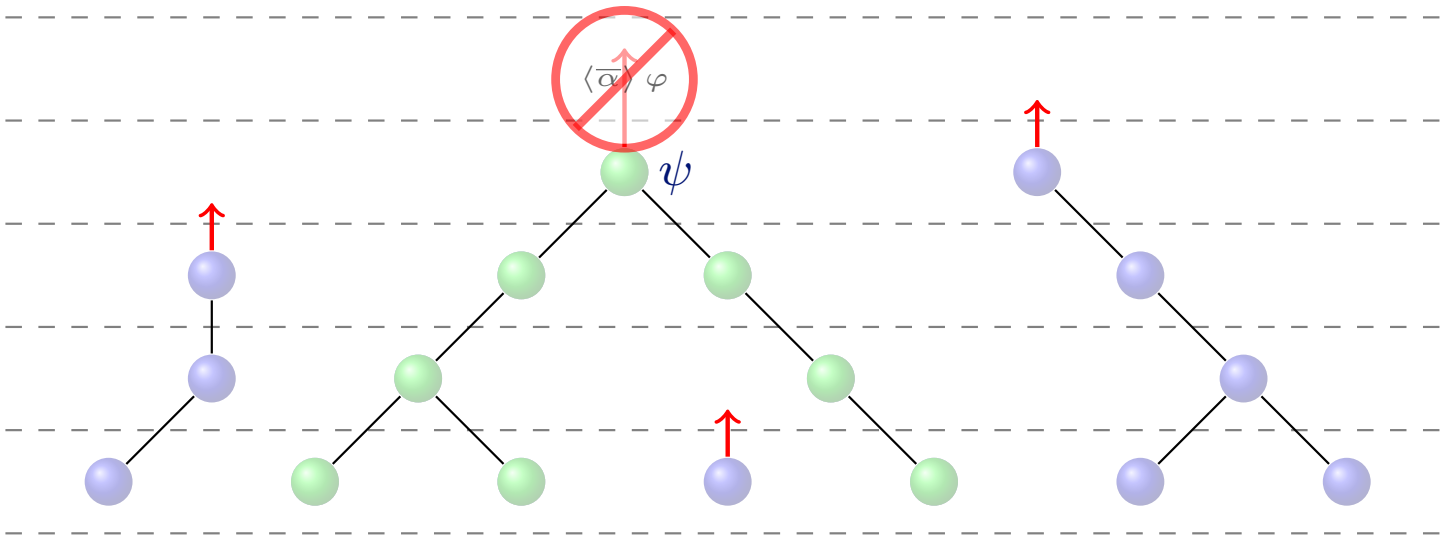
Satisfiability-Testing Algorithm: Principles



Progressive bottom-up reasoning (partial satisfiability)

- $\langle\overline{\alpha}\rangle\,\varphi$ are left unproved until a parent is connected

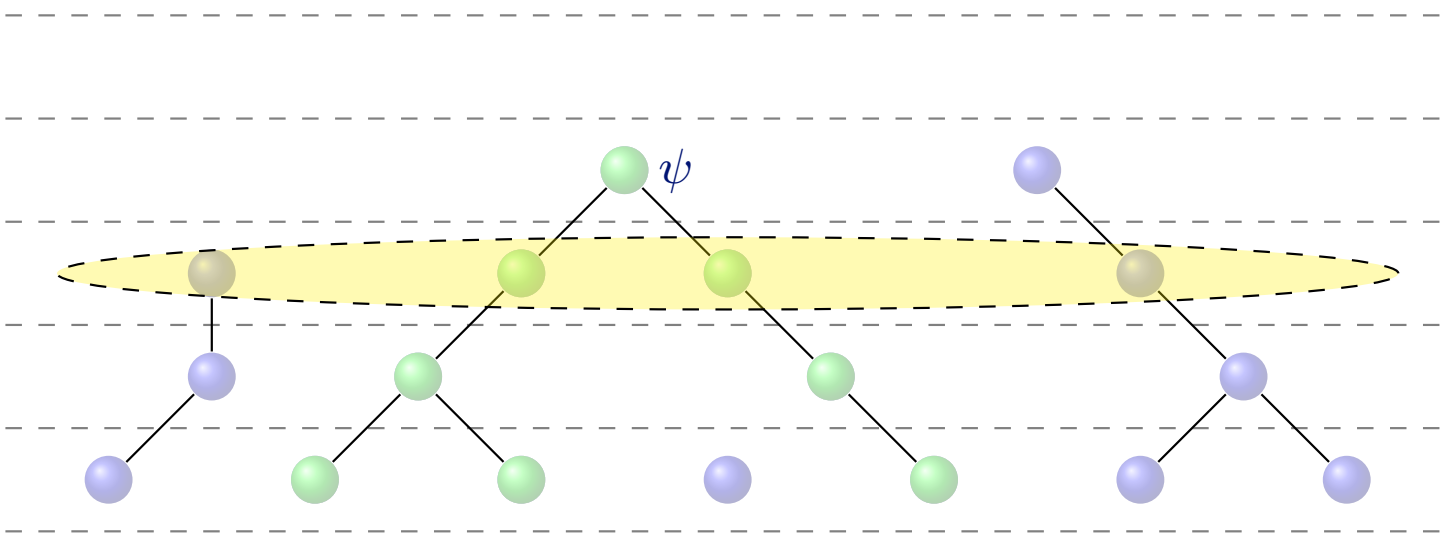
Satisfiability-Testing Algorithm: Principles



Termination

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

Satisfiability-Testing Algorithm: Principles



Implementation techniques

- Crucial optimization: symbolic representation

Correctness & Complexity

Theorem

The satisfiability problem for a formula $\psi \in \mathcal{L}_\mu$ is decidable in time $2^{O(n)}$ where $n = |Lean(\psi)|$.

Theorem

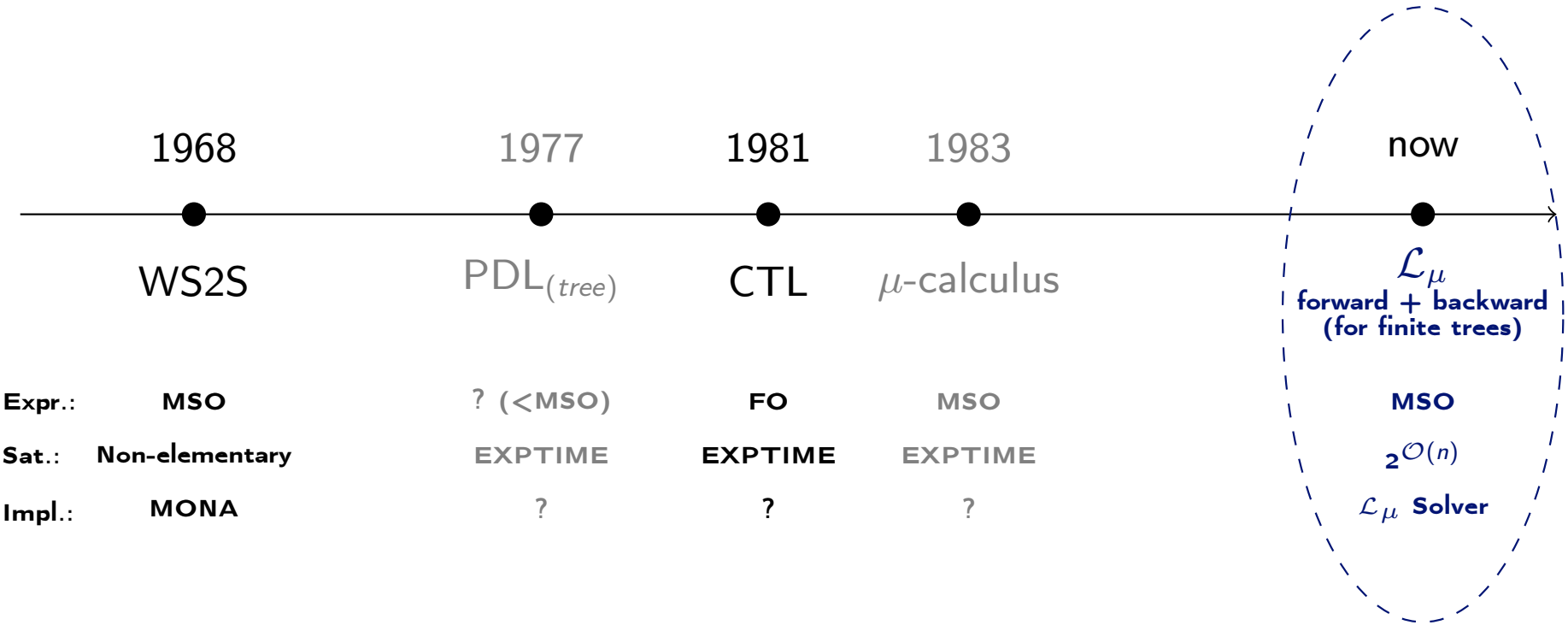
Translations of XPath and XML types into the logic are linear.

Corollary

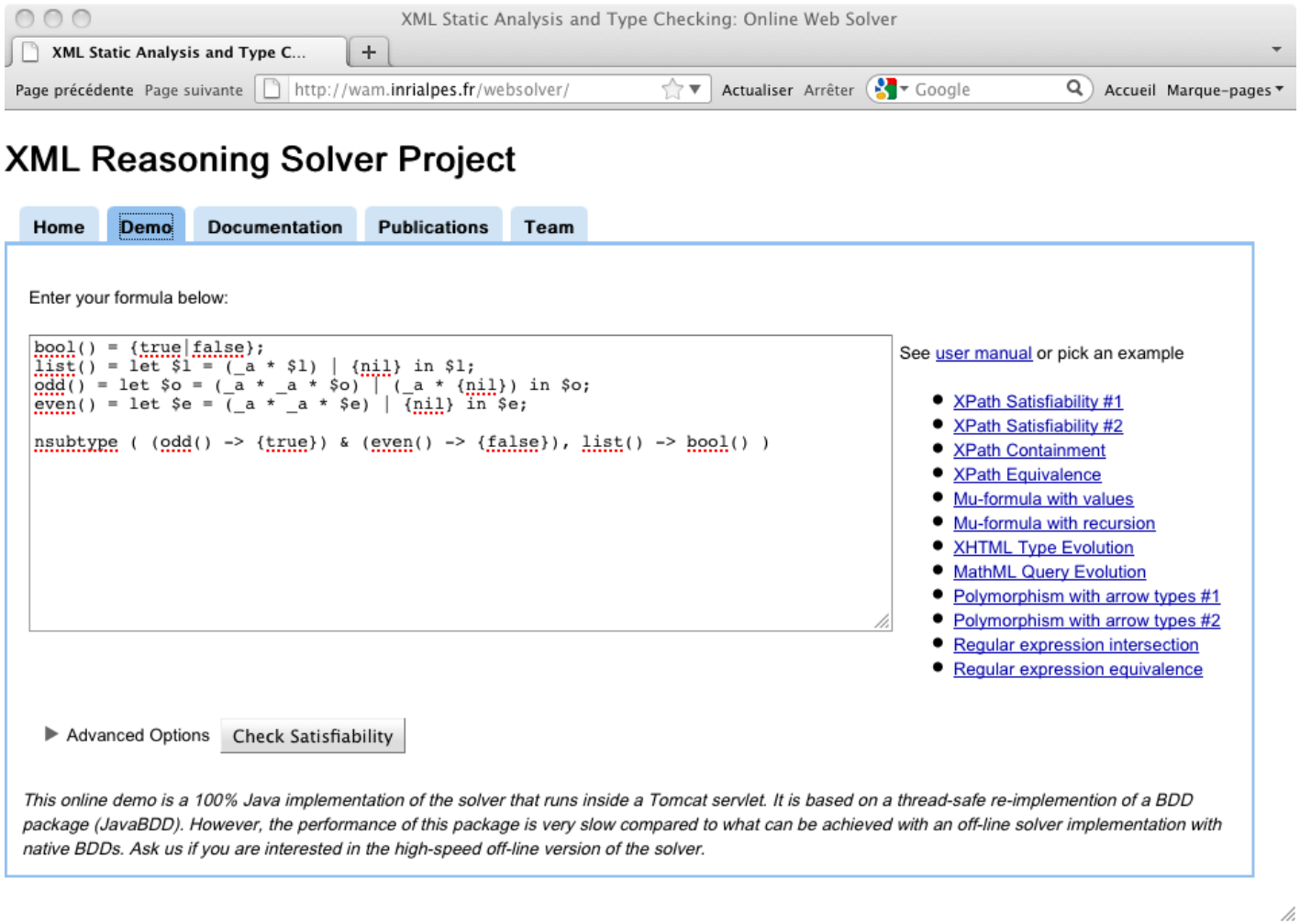
Decision problems involving XPath and types (e.g. typing, containment, emptiness, equivalence) can be decided in time $2^{O(n)}$.

Tree Logics: An Overview

- On the theoretical side: $\mathcal{L}_\mu$ offers an interesting expressivity, succinctness, optimal complexity bound



Prototype online*: http://tyrex.inria.fr/websolver



* or offline if performance is critical: the offline version is faster (native BDD library, further optimizations like compression of symbols) + latest advances

Overview of Some Experiments with Our Static Analyzers

Sample Problem	Lean Size	Time
Simple RE intersection & equivalence	30	15 ms
Query containment $q \subseteq q'$ (XPath)	50	50 ms
Query satisfiability with constraints (e.g. SMIL 1.0)	90	350 ms
Subtyping with rich types	60	70 ms
Schema evolution (moderate: e.g. XHTML-Basic)	170	2.5 s
Schema evolution (large: e.g. MathML)	290	8 s
Schema evolution (huge & complex, with attributes)	550	? | 27 s
Analysis of style sheets (many such calls)	60	40 ms
Precise typing for XQuery (many such calls)	70	35 ms

Table: Kinds of decision problems and corresponding figures.

For some test, size of the Lean is 550. The search space is $2^{550} \approx 10^{165}$... more than the square number of atoms in the universe 10^{80}

Some Applications

- University of Washington (USA): query **intersection** in analysing web page scripting
- University of Maryland (USA): analysing **access control** policies (e.g. XACML)
- University of Edinburgh (UK): query containment for **XML databases**
- Institute of CS-FORTH (Greece): access control system for documents
- University of British Columbia (Canada): software engineering for the cloud
- Universität Stuttgart (Germany): analysis of **BPEL** data flows

A Domain with Broad Applications

Advances in this domain $\Leftrightarrow$ Advances in core computer science

Applications almost everywhere:

- Next-Generation Programming Languages
- NoSQL Databases
- Modern File Systems
- Structured and Compound Documents
- Web Engineering
- Semantic Web
- Data Structure/Program Analysis and Verification
- Optimizing Compilers for Big Data Infrastructures
- ...

The TyReX Research Team

http://tyrex.inria.fr pierre.geneves@cnrs.fr

Two References of the Team

Pierre Genevès, Nabil Layaïda, Alan Schmitt, Nils Gesbert.

Efficiently Deciding μ-Calculus with Converse over Finite Trees.

ACM Transactions on Computational Logic 16(2): 16:1-16:41 (2015)

Pierre Genevès and Nils Gesbert.

XQuery and Static Typing: Tackling the Problem of Backward Axes.

In ICFP'15: Proceedings of the 2015 ACM SIGPLAN Conference on Functional Programming, 2015.

More Related Applications

Whose investigation was successful using this approach:

- Query containment for XPath with constraints [PLDI'07, ICDE'10]
- Modeling counting and shuffle/interleaving operators [IJCAI'11]
- Dead code analysis for XQuery [ICSE'10, ICSE'11]
- Impact of schema evolutions on queries [ICFP'09, TOIT'11]
- Deciding subtyping for rich type algebras with functions/polymorphism [ICFP'11]
- Containment for SPARQL queries under constraints [AAAI'12, IJCAR'12]
- Query update independence analysis [DocEng'12]
- Static verification of layouts: bug detection for CSS Stylesheets [WWW'12]