A Taste of Research...

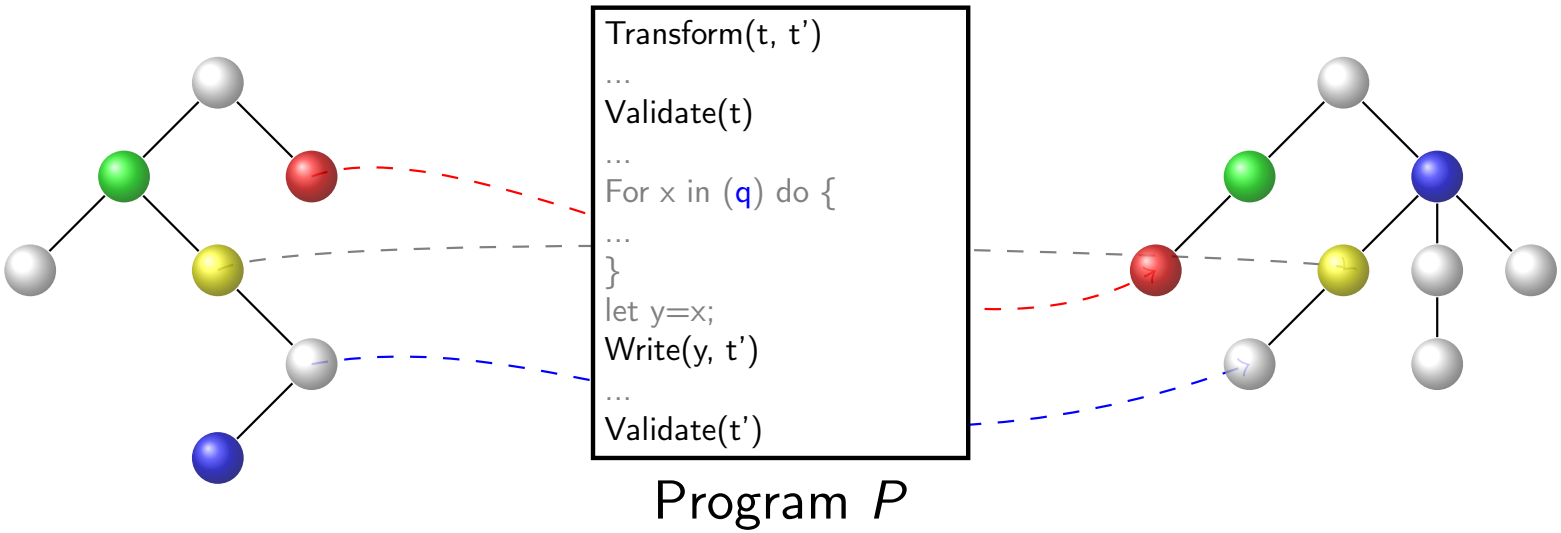
Pierre Genevès
CNRS

University of Grenoble, 2011–2012

Some Research Objectives

- Design and develop languages/algorithms/programming techniques on solid foundations for answering new needs required by the latest evolutions of the World Wide Web...
- Understand the expressive power and complexity of XML languages
- Identify interesting fragments/extensions
- Contribute to the standardization effort...
- ... and to the next generation Web!

XML Processing Programs



3 Essential Tasks

- Validation: check that an XML document is valid w.r.t. a given type
- Navigation/Extraction: select a set of nodes (q: XPath expression)
- Transformation: build a new document from an existing one

A Line of Challenges for the Years to Come

"Data manipulations should be safe and efficient!"

- Ensure that programs safely and efficiently manipulate XML data (through type systems, static analyzers and optimizing compilers)
- Introduce XML as a first-class citizen in programming languages (statically typed XML processing)

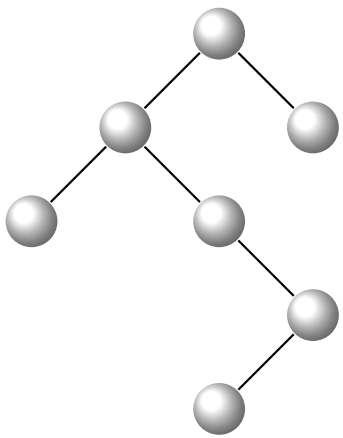
Key (and hard) problem

- Reasoning with XML types and (XPath) queries

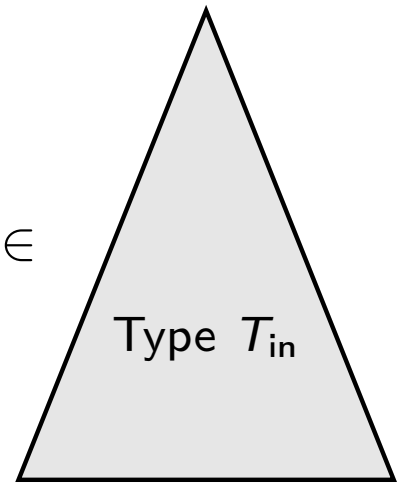
Possible Approach (WAM)

- ✗ Do not design a n^{th} new programming language for XML processing
- ✓ Ensure those which are used for this purpose are safe and efficient (whatever the programming language family is)
- ✓ Design static analysis methods for XML processing

Static Analysis for XML Processing: Examples



$\in$

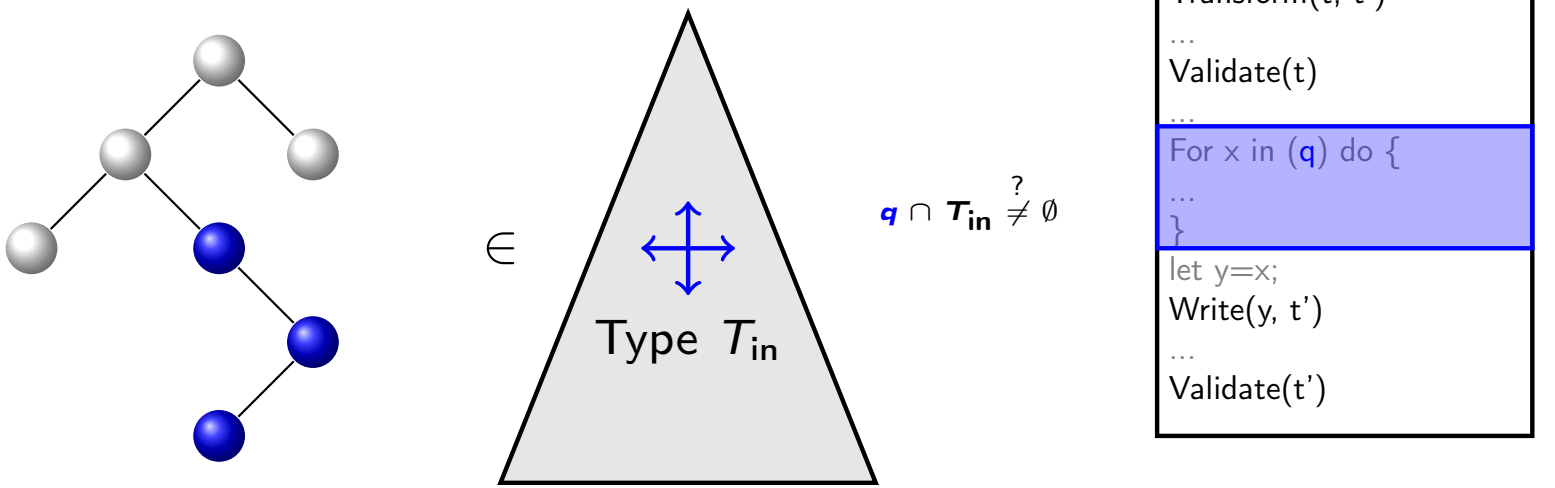



```
Transform(t, t')
...
Validate(t)
...
For x in (q) do {
...
}
let y=x;
Write(y, t')
...
Validate(t')
```

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

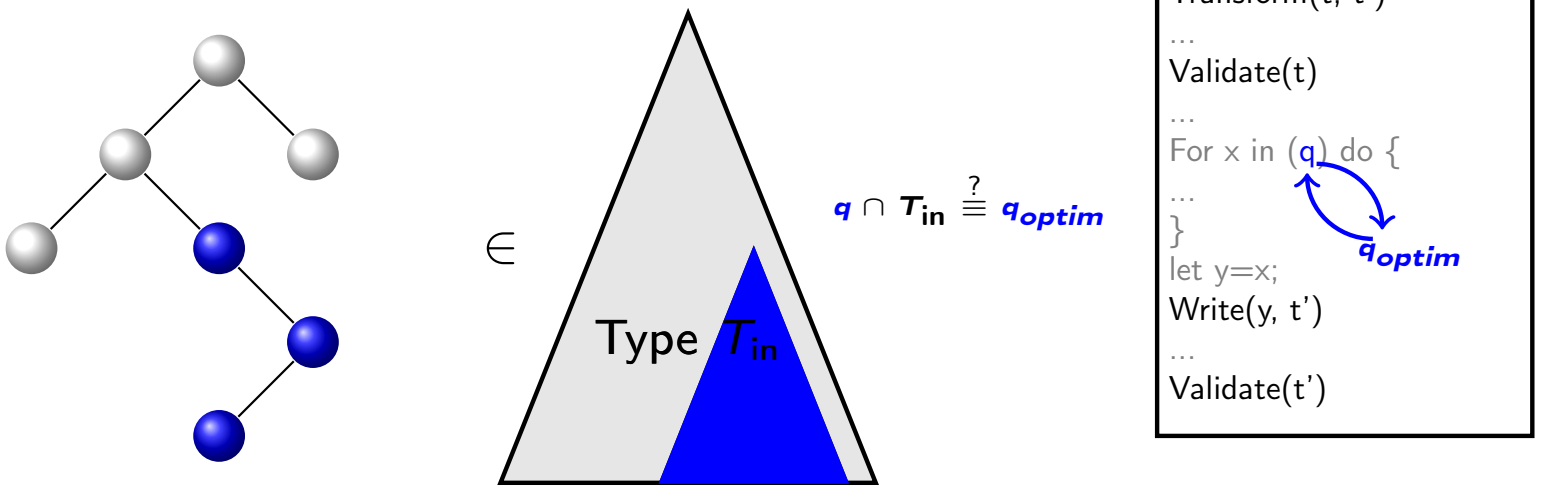
- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Static Analysis for XML Processing: Examples



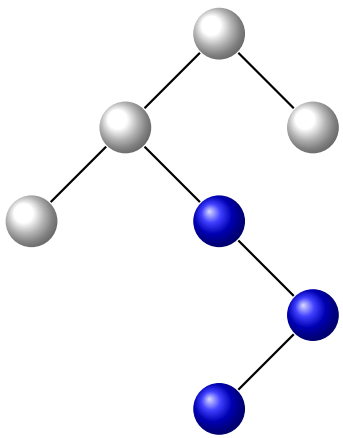
- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Static Analysis for XML Processing: Examples

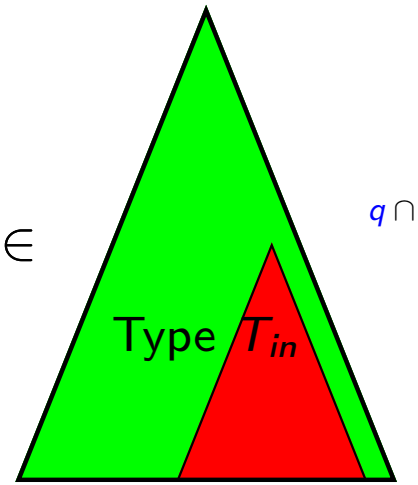


- Detecting errors (invalid output, empty queries)
- Optimization program execution (avoiding language overuse, tree size)
- Checking programs properties (security holes, termination, proofs...)

Static Analysis for XML Processing: Examples



$\in$

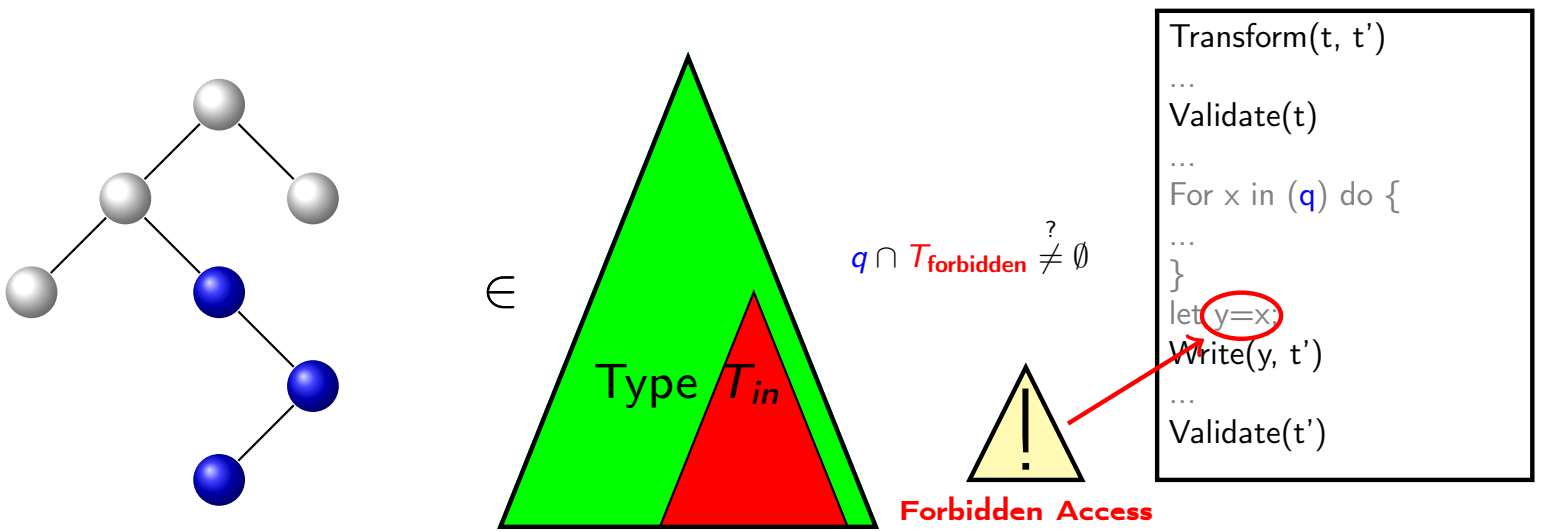


$q \cap T_{\text{forbidden}} \stackrel{?}{\neq} \emptyset$

```
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...)

Static Analysis for XML Processing: Examples



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

Difficulties and Major Challenges

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

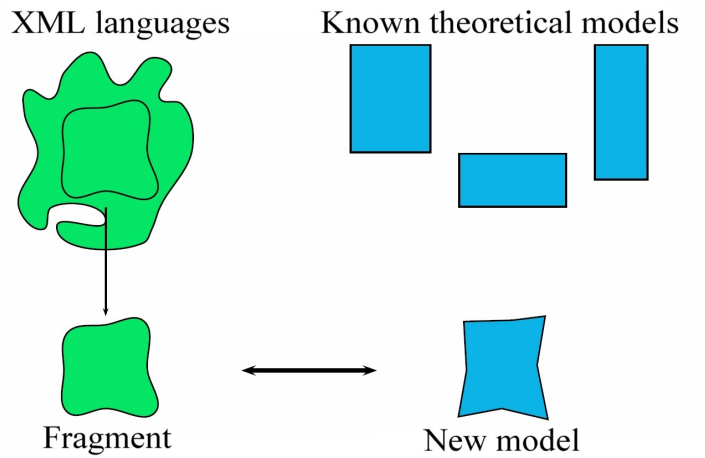
- ✓ capture regular tree types
- × how to capture $\mathcal{L}_{\text{XPath}}$?

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)

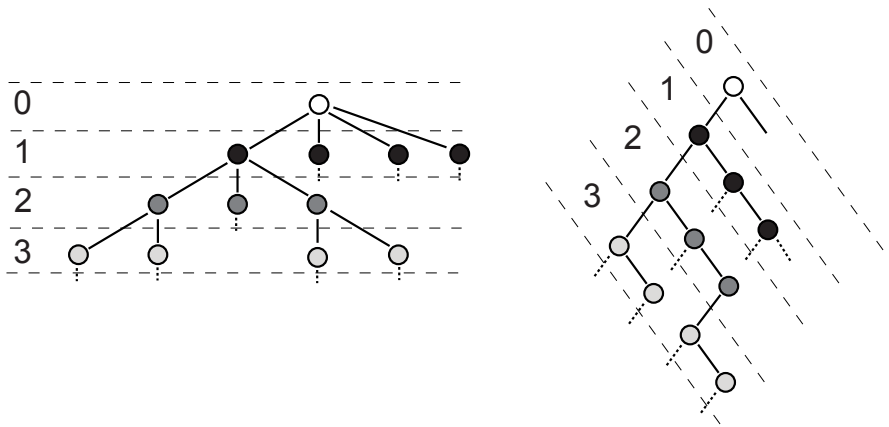
Outline

Some Recent Results (WAM) [?]

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

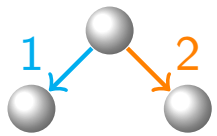


General Encoding

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

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

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



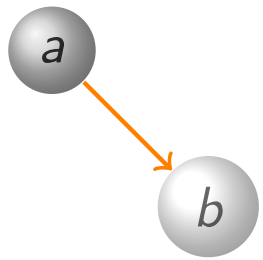
$\mathcal{L}_\mu \ni \varphi$	$::=$		formula
		$\top$	true
	$\mid$	$\sigma \mid \neg\sigma$	atomic prop (negated)
	$\mid$	$\varphi \vee \varphi \mid \varphi \wedge \varphi$	disjunction (conjunction)
	$\mid$	$\langle\alpha\rangle\,\varphi \mid \neg\langle\alpha\rangle\,\top$	existential (negated)
	$\mid$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

Sample Formula and Satisfying Tree

a

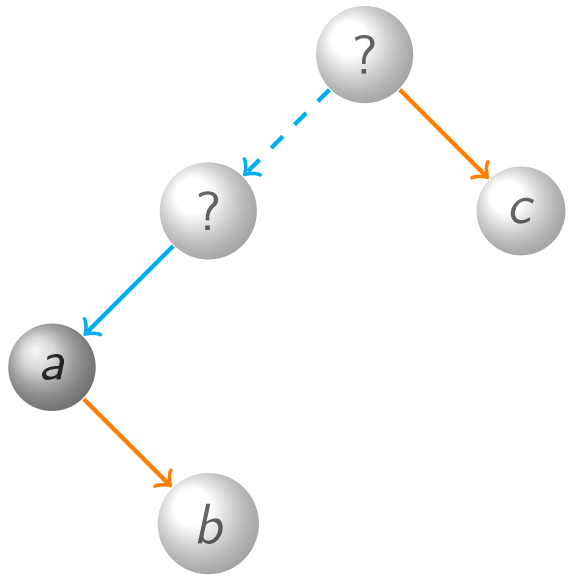
Sample Formula and Satisfying Tree

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



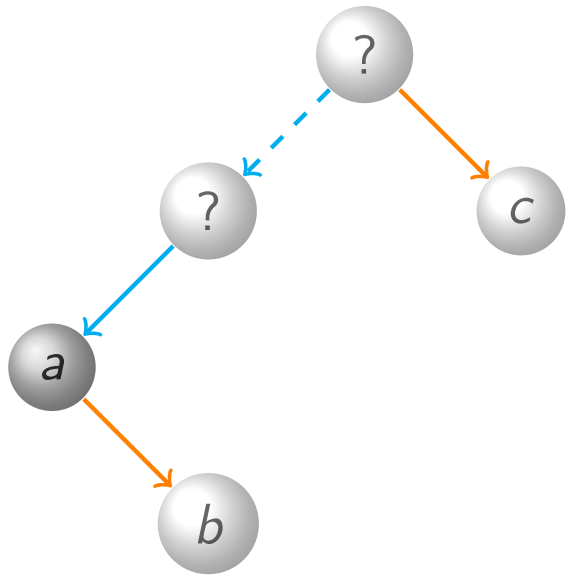
Sample Formula and Satisfying Tree

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



Sample Formula and Satisfying Tree

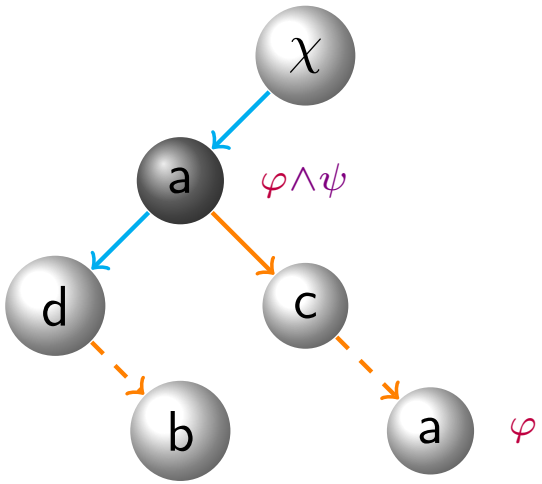
$$a \wedge \langle 2 \rangle\, b \wedge \mu X.\, \langle 2 \rangle\, c \vee \langle \overline{1} \rangle\, X$$



- 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

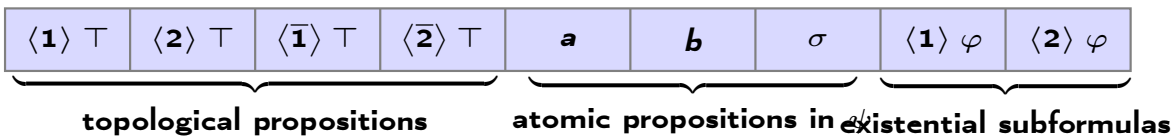
- Search for a proof tree
- Build the proof bottom up: if ψ holds then it is necessarily somewhere up

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

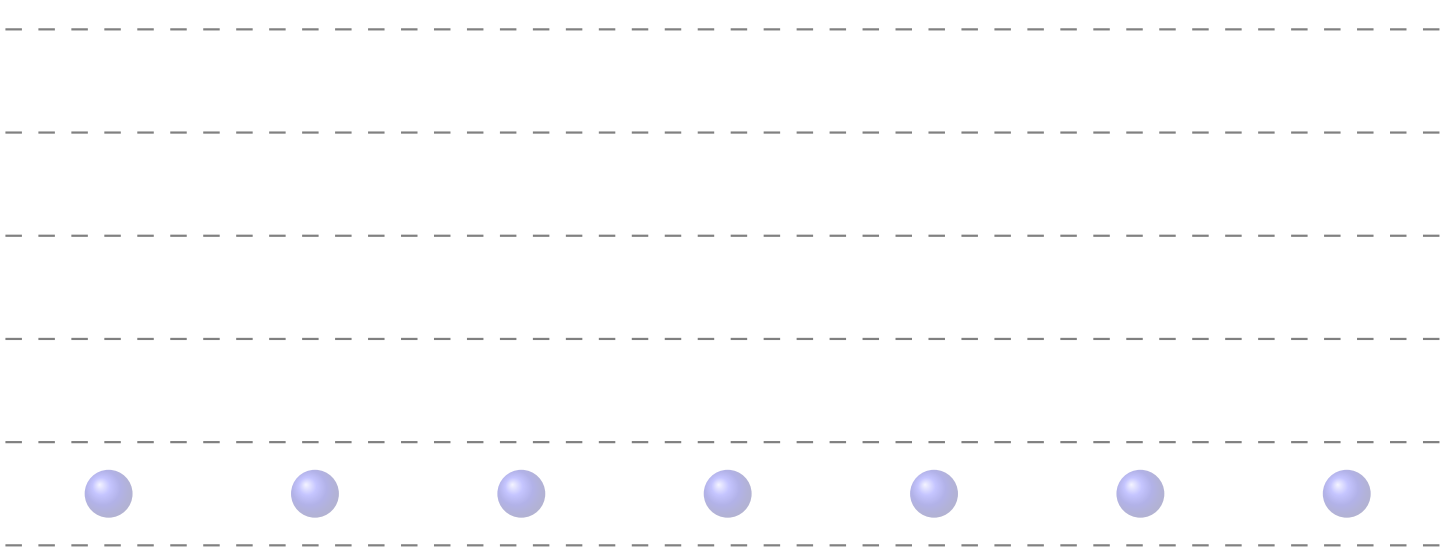
- With some additional constraints, e.g. $\neg \langle \overline{1} \rangle \top \vee \neg \langle \overline{2} \rangle \top$

Satisfiability-Testing Algorithm: Principles

Bottom-up construction of proof tree

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

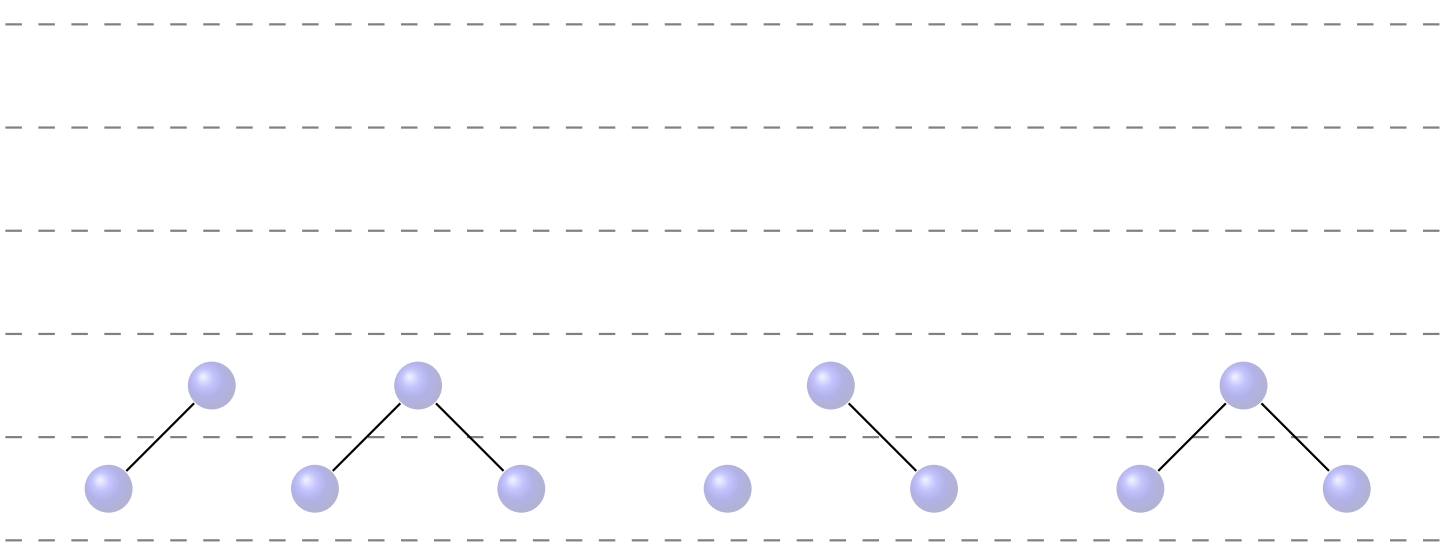
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

- Step 1: all possible leaves are added

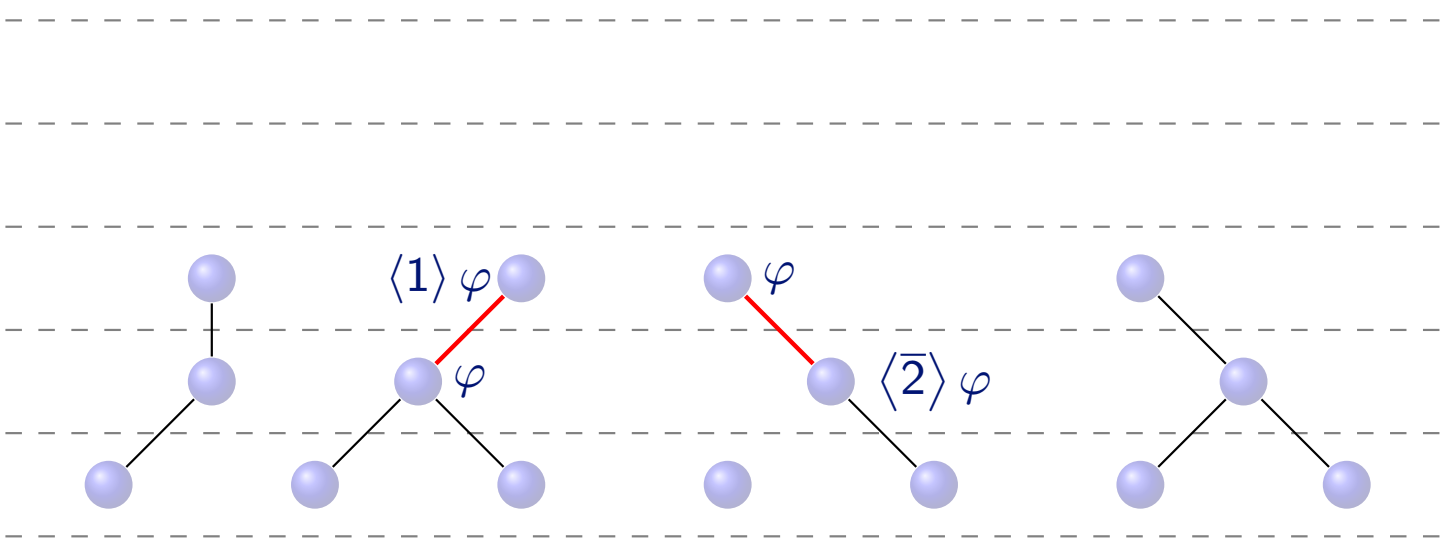
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

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

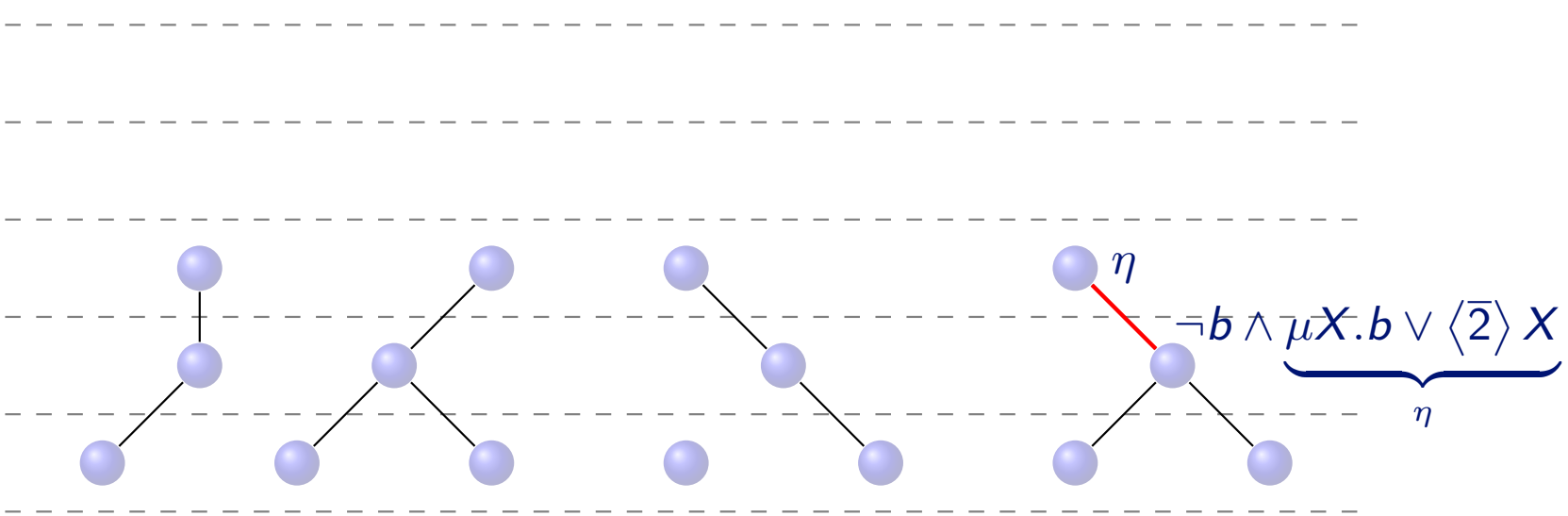
Satisfiability-Testing Algorithm: Principles



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

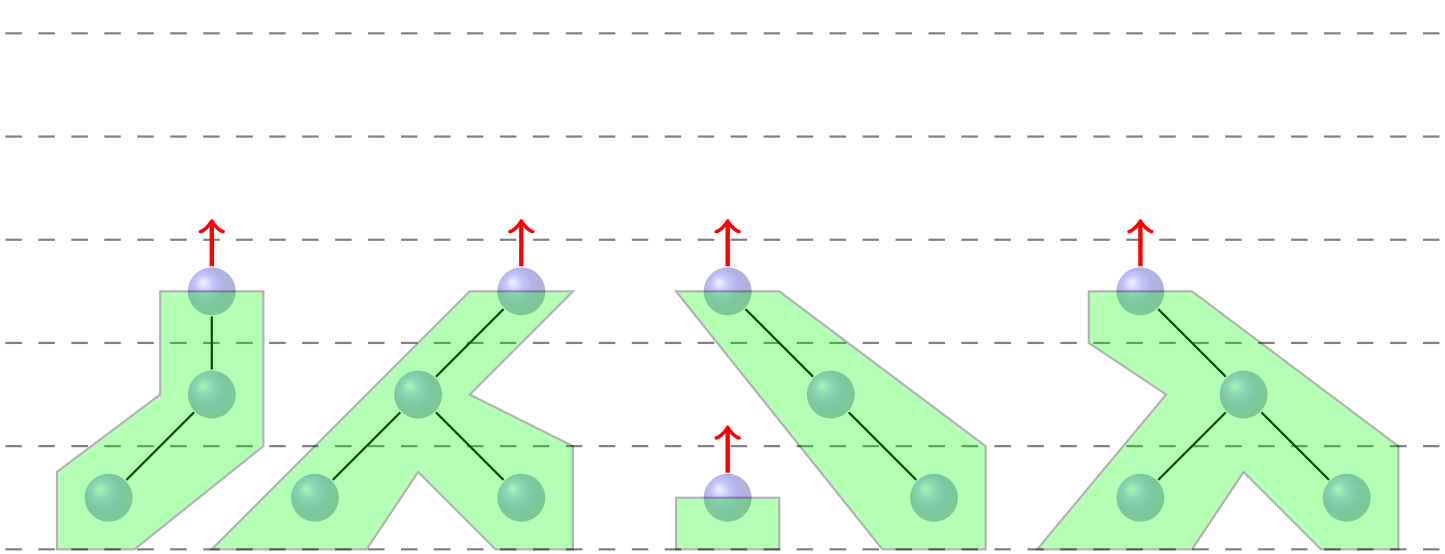
Satisfiability-Testing Algorithm: Principles



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

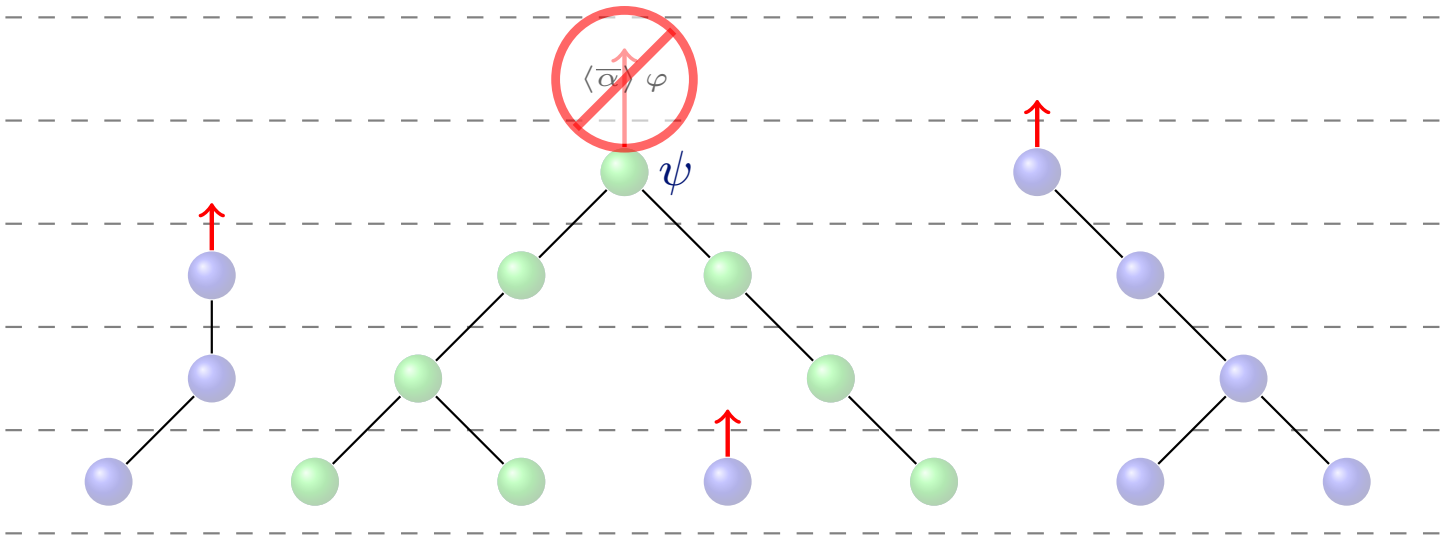
Satisfiability-Testing Algorithm: Principles



Progressive bottom-up reasoning (partial satisfiability)

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

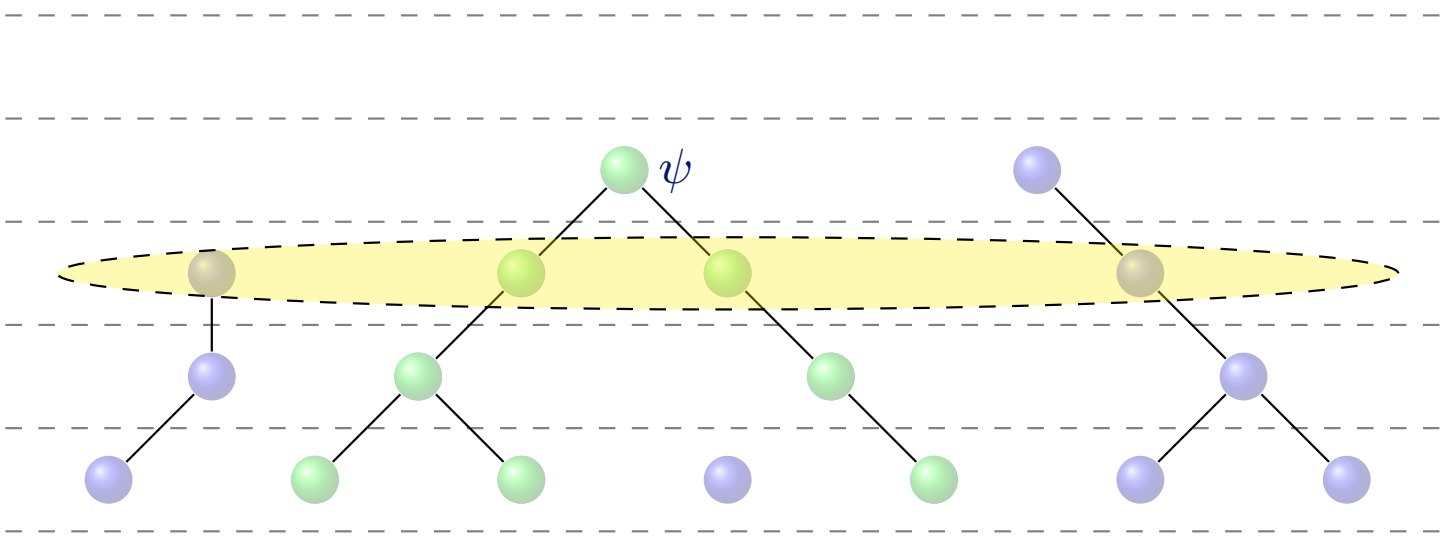
Satisfiability-Testing Algorithm: Principles



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 $2^{O(n)}$.

System fully implemented: solver + XPath & XML types compilers [PLDI'07]

Overview of Experiments

DTD	Symbols	Binary type variables
SMIL 1.0	19	11
XHTML 1.0 Strict	77	325

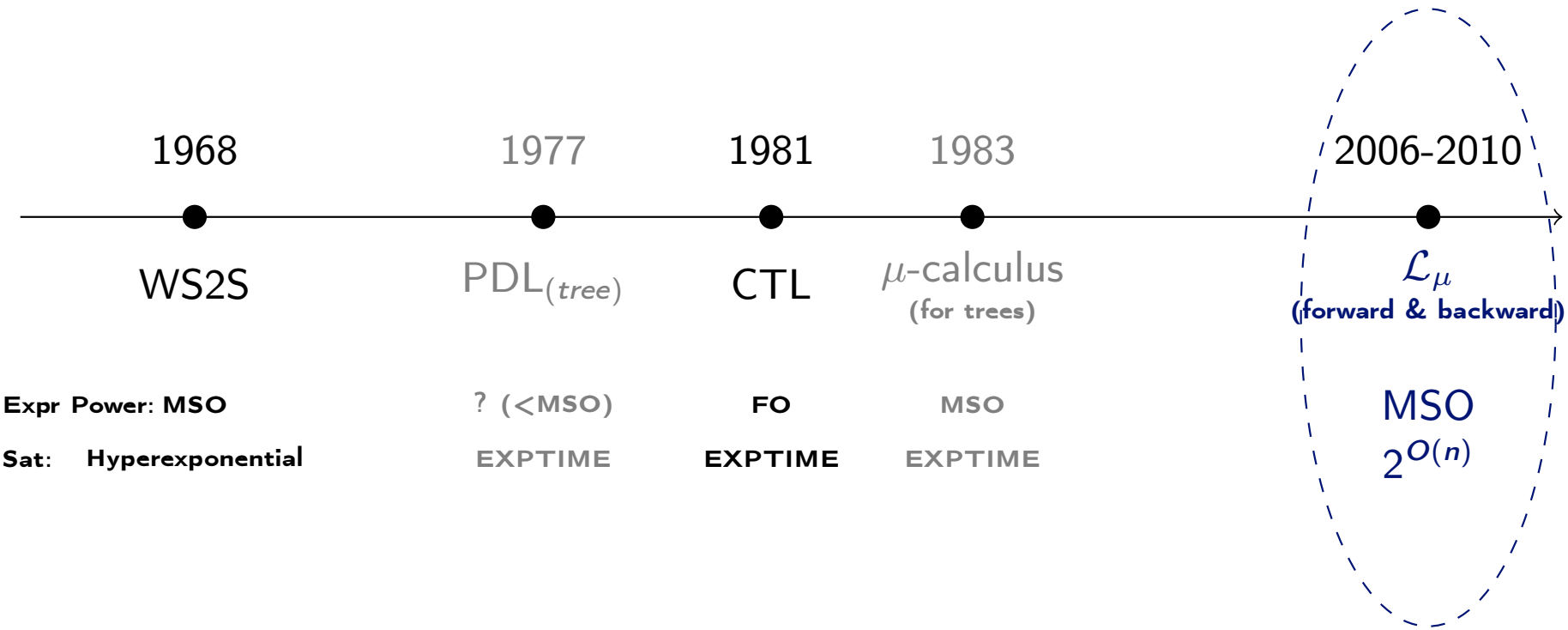
Table: Types used in experiments.

XPath decision problem	XML type	Time (ms)
$e_1 \subseteq e_2$ and $e_2 \not\subseteq e_1$	none	353
$e_4 \subseteq e_3$ and $e_4 \subseteq e_3$	none	45
$e_6 \subseteq e_5$ and $e_5 \not\subseteq e_6$	none	41
e_7 is satisfiable	SMIL 1.0	157
e_8 is satisfiable	XHTML 1.0	2630
$e_9 \subseteq (e_{10} \cup e_{11} \cup e_{12})$	XHTML 1.0	2872

Table: Some decision problems and corresponding results.

For the last 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}

Tree Logics: an Overview



A Domain with Broad Applications

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

Applications almost everywhere:

- Next-Generation Programming Languages
- XML Databases
- Modern File Systems
- Structured and Compound Documents
- Web Engineering
- Semantic Web
- Data Structure/Program Analysis and Verification
- ...

Master Internship / Ph.D. Opportunities @ INRIA

http://wam.inrialpes.fr - pierre.geneves@inria.fr


Genevès, P., Layaïda, N., and Schmitt, A. (2007).
Efficient static analysis of XML paths and types.
In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 342–351, New York, NY, USA. ACM Press.