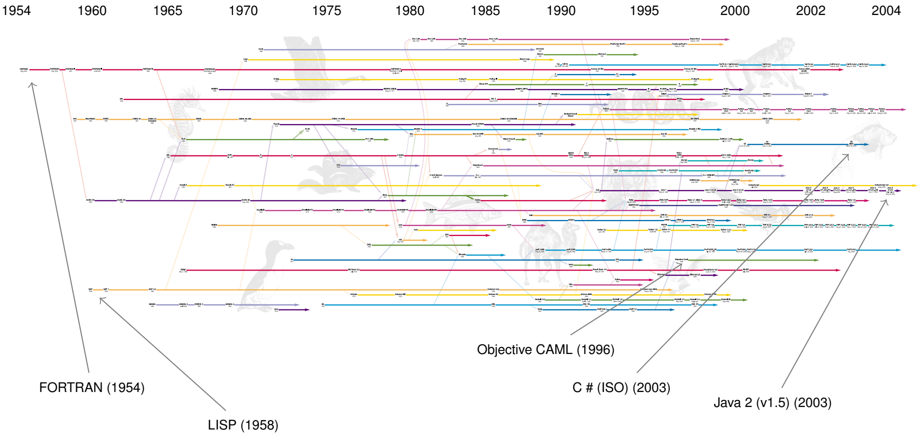
When the Web Meets Programming Languages and Databases: Foundations of XML

Pierre Genevès

CNRS – LIG – WAM

May 16th, 2008

History of Programming Languages

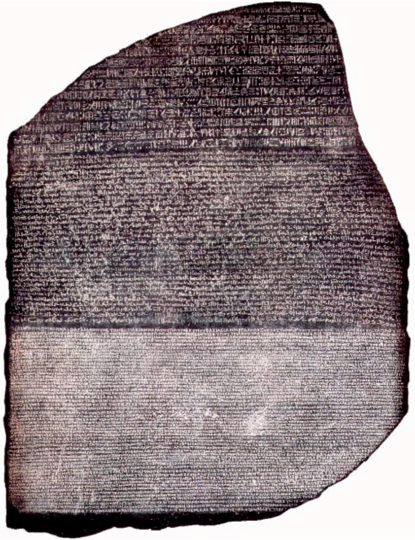


What about Data?

- Often, data is more important than programs (e.g. banks, aeronautical technical documentation, ...)
- How to ensure long-term access to data?

What about Data?

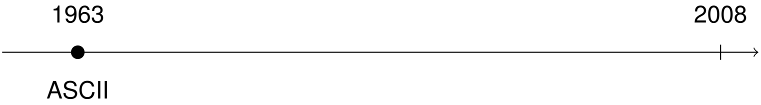
- Often, data is more important than programs (e.g. banks, aeronautical technical documentation, ...)
- How to ensure long-term access to data?
- A quite old problem...
- Can we really do better with computers?



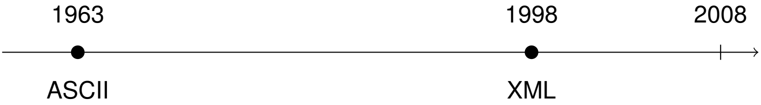
La pierre de Rosette.

What has not changed for 50 years in Computer Science?

Standards for Data Representation



Standards for Data Representation



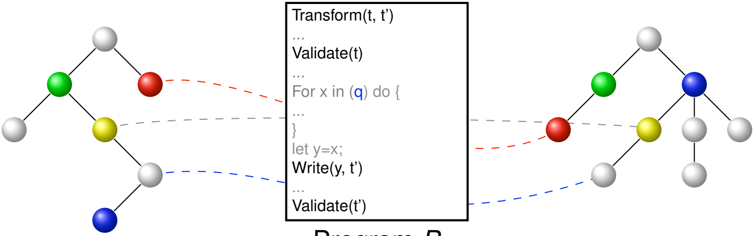
- Before: file format tied to a processor (due to processor-specific instructions)
- After: markup language for describing (structured) data in itself (independently from processors)

Back in 2008

Now

- XML is the Lingua franca for communicating on the web (documents, mobiles data, financial data, molecules, architectures ...)
- Two key notions:
  - XML Types: define constraints on children and siblings of nodes using regular expressions
  - XML Queries: expressions for selecting a set of matching nodes (XPath standard)

Processing XML Documents



Program P

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

Major Challenge for the Years to Come

Data manipulations must be safe and efficient

- × Do not design a n^{th} new programming language for XML processing
- ✓ Ensure that those which are used for this purpose are safe and efficient (whatever the programming language family is)
- ✓ Introduce XML as a first-class citizen in programming languages

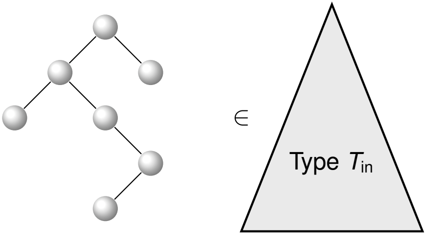
Key (and hard) problem

- Reasoning with XML types and (XPath) queries

Approach

- Design static analysis methods for XML processing

Static Analysis for XML Processing: Examples

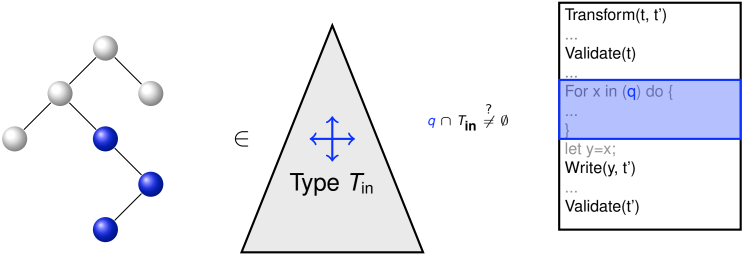



```
Transform(t, t')
...
Validate(t)
...
For x in (q) do {
...
}
let y=x;
Write(y, t')
...
Validate(t')
```

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

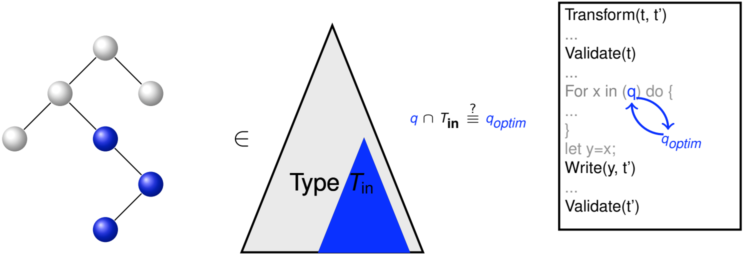
- Errors are very difficult to detect (invalid output, empty queries)
- Huge performance problems at execution (language overuse, tree size)
- Difficult to check properties on programs (security holes, termination...)

Static Analysis for XML Processing: Examples



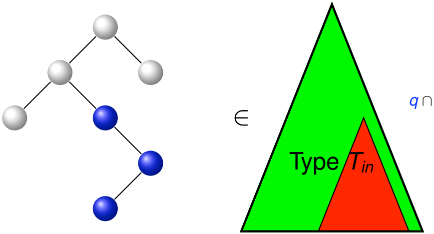
- Errors are very difficult to detect (invalid output, empty queries)
- Huge performance problems at execution (language overuse, tree size)
- Difficult to check properties on programs (security holes, termination...)

Static Analysis for XML Processing: Examples



- Errors are very difficult to detect (invalid output, empty queries)
- Huge performance problems at execution (language overuse, tree size)
- Difficult to check properties on programs (security holes, termination...)

Static Analysis for XML Processing: Examples

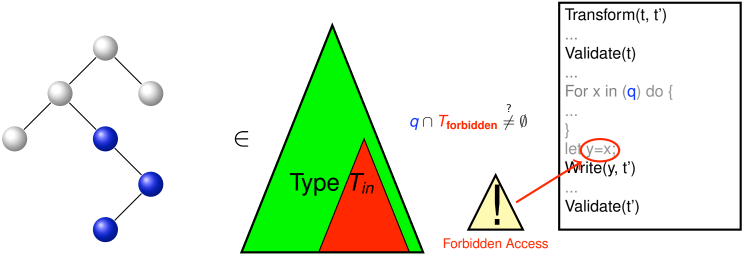


$q \cap T_{\textbf{forbidden}} \overset{?}{\neq} \emptyset$

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

- Errors are very difficult to detect (invalid output, empty queries)
- Huge performance problems at execution (language overuse, tree size)
- Difficult to check properties on programs (security holes, termination...)

Static Analysis for XML Processing: Examples



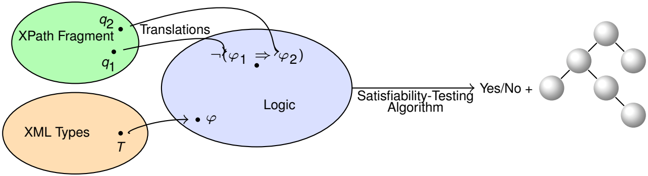
- Errors are very difficult to detect (invalid output, empty queries)
- Huge performance problems at execution (language overuse, tree size)
- Difficult to check properties on programs (security holes, termination...)

Outline

1. The logical approach for static analysis
2. A logic of finite ordered trees for XML
3. Satisfiability-testing algorithm

The Logical Approach for XML Types/XPath Analysis

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



Advantages

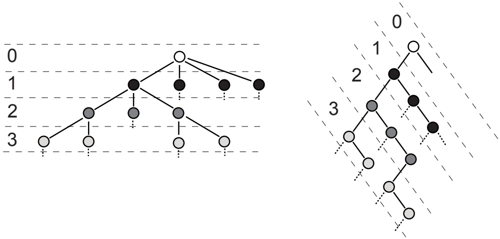
- Solve problems that are boolean combination of other problems
- Provide formal proofs of safety properties and bugs
- Can be used in synthesis: query optimization, counter examples, etc.

Requirements for the Logic

1. Expressive enough to capture XPath and XML types but succinct
2. Best complexity
3. Nice algorithmic properties (not always worst case)

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

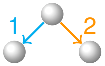


General Encoding

- 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 ($\overline{\overline{\alpha}} = \alpha$)



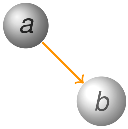
$\mathcal{L}_\mu \ni \varphi, \psi \quad ::=$		formula
	$\top$	true
$\mid$	$\sigma \quad \mid \quad \neg\sigma$	atomic prop (negated)
$\mid$	Ⓢ $\quad \mid \quad \neg$Ⓢ	context (negated)
$\mid$	$\varphi \vee \psi \quad \mid \quad \varphi \wedge \psi$	disjunction (conjunction)
$\mid$	$\langle\alpha\rangle\,\varphi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

Sample Formula and Satisfying Tree

a

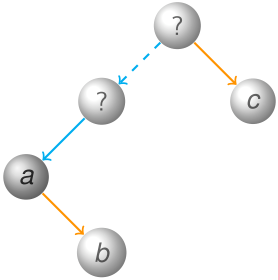
Sample Formula and Satisfying Tree

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



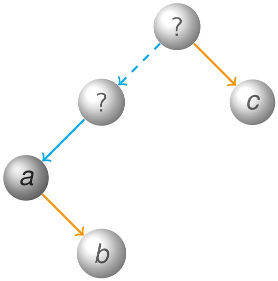
Sample Formula and Satisfying Tree

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



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

Outline

1. The logical approach for static analysis
2. A logic of finite ordered trees for XML
3. Satisfiability-testing algorithm

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) :

topological propositions				atomic propositions in ψ			existential subformulas	
$\langle 1 \rangle \top$	$\langle 2 \rangle \top$	$\langle \bar{1} \rangle \top$	$\langle \bar{2} \rangle \top$	a	b	σ	$\langle 1 \rangle \varphi$	$\langle 2 \rangle \varphi$

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

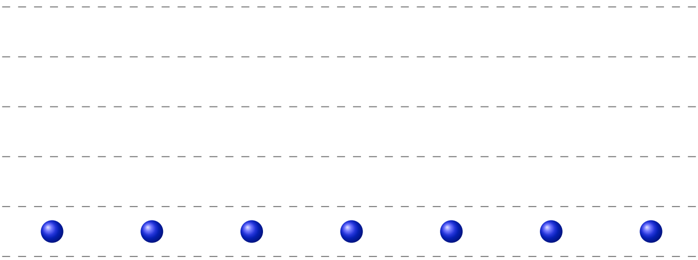
- With some additional constraints, e.g. $\neg \langle \bar{1} \rangle \top \vee \neg \langle \bar{2} \rangle \top$

Satisfiability-Testing Algorithm: Principles

Bottom-up construction of proof tree

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

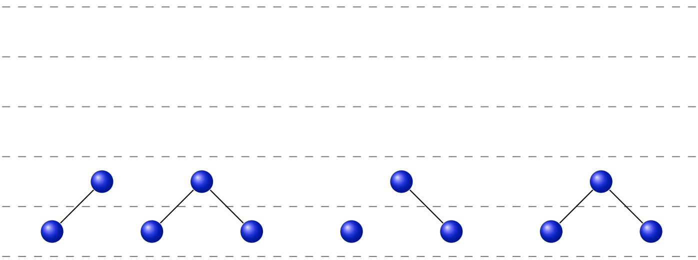
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

- Step 1: all possible leaves are added

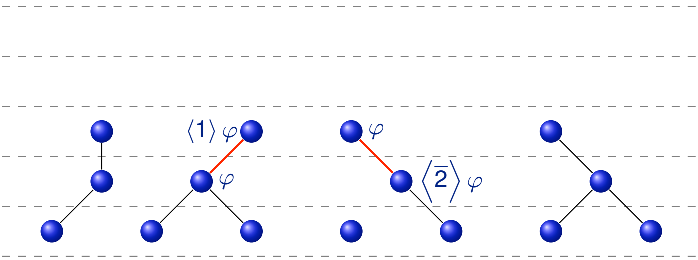
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

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

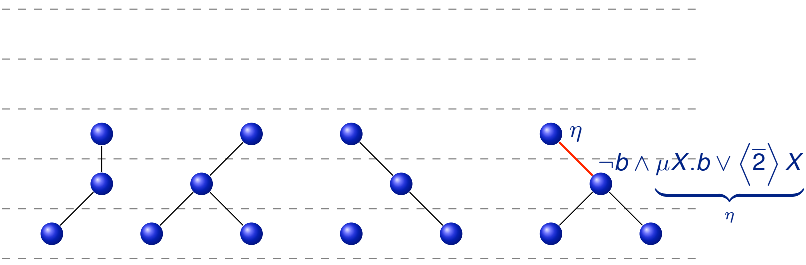
Satisfiability-Testing Algorithm: Principles



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

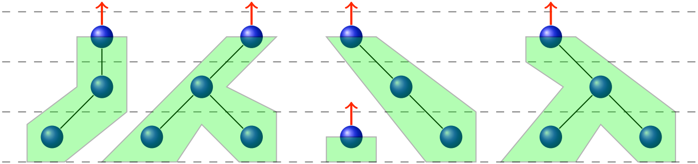
Satisfiability-Testing Algorithm: Principles



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

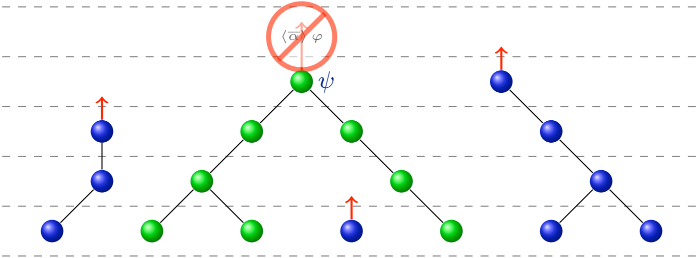
Satisfiability-Testing Algorithm: Principles



Progressive bottom-up reasoning (partial satisfiability)

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

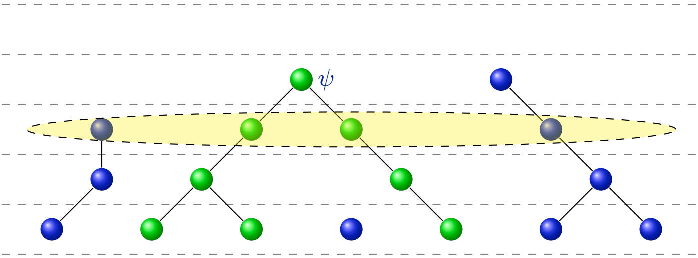
Satisfiability-Testing Algorithm: Principles



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

	1968	1977	1981	1983	2006
	WS2S	$\text{PDL}_{(tree)}$	CTL	μ-calculus (for trees)	$\mathcal{L}_\mu$ (forward & backward)
Expr Power:	MSO	? (<MSO)	FO	MSO	MSO
Sat:	Hyperexponential	EXPTIME	EXPTIME	EXPTIME	$2^{O(n)}$

Future Works

Extending the tree logic

- Decidable counting constraints and data-value comparisons
- Higher order XML types (web services)
- Logic for graphs (semantic web)

Binary Trees... (Desert of California).

Some applications

- XML code optimization / security
- XML databases
- C/Java code analysis (Bohne, PALE)
- Typing of (reconfigurable) component based languages (FScript/Sardes)
- ...