A Tree Logic...

... and an Application for the Analysis of Cascading Style Sheets

Pierre Genevès

CNRS – Tyrex team
pierre.geneves@inria.fr

Toccata seminar, LRI – Feb. 22^{nd}, 2013

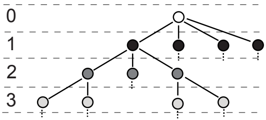
Outline

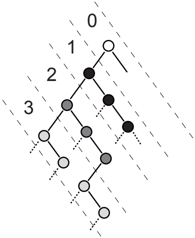
1. Insights on the $\mathcal{L}_\mu$ Tree Logic
2. Overview of Perspectives and Applications
3. Zoom on the Analysis of CSS

Data Model for the Logic

Trees: the logic was originally designed for XML trees

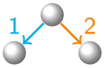
- Specifically: finite binary labeled trees
- They model finite ordered unranked labeled trees wlog
- Bijective encoding of unranked trees as binary trees:





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

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



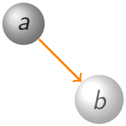
$$
\begin{array}{rclll}
\mathcal{L}_\mu \ni \varphi, \psi & ::= & & & \text{formula} \\
& & \top & & \quad \text{true} \\
& | & p \quad | \quad \neg p & & \quad \text{atomic prop (negated)} \\
& | & n \quad | \quad \neg n & & \quad \text{nominal (negated)} \\
& | & \varphi \vee \psi \quad | \quad \varphi \wedge \psi & & \quad \text{disjunction (conjunction)} \\
& | & \langle \alpha \rangle \varphi \quad | \quad \neg \langle \alpha \rangle \top & & \quad \text{existential (negated)} \\
& | & \mu X.\varphi & & \quad \text{unary fixpoint (finite recursion)} \\
& | & \mu \overline{X_i.\varphi_i} \text{ in } \psi & & \quad n\text{-ary fixpoint}
\end{array}
$$

Sample Formula and Satisfying Tree

a

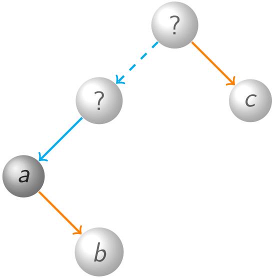
Sample Formula and Satisfying Tree

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



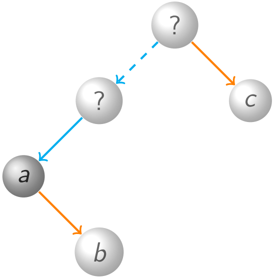
Sample Formula and Satisfying Tree

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



Sample Formula and Satisfying Tree

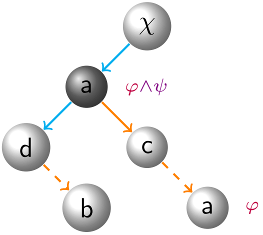
$$a \wedge \langle 2 \rangle\, b \wedge \mu X.\, \langle 2 \rangle\, c \vee \langle \overline{1} \rangle\, X$$



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

✓ Interesting balance between succinctness and expressive power: XPath, CSS selectors, and XML types can be translated into the logic, linearly

Example: Translation of an XPath Expression into $\mathcal{L}_\mu$



- Formula holds at selected nodes
- $\mu Z.\varphi$: finite recursion
- Converse programs are crucial
- More generally, we have a compiler:
  - $t_{\mathsf{xpath}}(e, \chi) : \mathcal{L}_{\mathsf{XPath}} \times \mathcal{L}_\mu \to \mathcal{L}_\mu$
  - χ is the latest navigation step
  - initially, $\chi = \neg \langle \overline{1} \rangle \top \wedge \neg \langle \overline{2} \rangle \top$ for absolute expressions

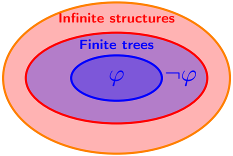
Translated query: child::a [child::b]

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

$\mathcal{L}_\mu$ Closure under Negation

Cycle-freeness: A key property

- If both a program and its converse occur between a $\mu X.$ binder and X, formula has a cycle, e.g.: $\mu X. \langle \alpha \rangle X \vee \langle \overline{\alpha} \rangle X$
- Otherwise the formula is cycle-free
- in practice, most (all?) formulas are cycle-free (e.g. XPath translations are always cycle-free)



- Cycle-freeness of $\mathcal{L}_\mu$ implies closure under negation
  - The negation of finite recursion is finite recursion (see paper)
  - $\neg\varphi$ is easily (linearly) expressible in $\mathcal{L}_\mu$ for all $\varphi \in \mathcal{L}_\mu$
- Crucial for BC: implication (subtyping, containment tests...)
- Crucial for implementation

Deciding $\mathcal{L}_\mu$ Satisfiability

Is a formula $\psi \in \mathcal{L}_\mu$ satisfiable?

- Given ψ, determine whether there exists a finite tree that satisfies ψ
- Validity: test $\neg\psi$

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)

$$\mathrm{Lean}(\psi): \underbrace{\boxed{\langle 1\rangle \top} \boxed{\langle 2\rangle \top} \boxed{\langle \bar{1}\rangle \top} \boxed{\langle \bar{2}\rangle \top}}_{\text{topological propositions}} \underbrace{\boxed{a} \boxed{b} \boxed{\sigma}}_{\text{atomic propositions in } \psi} \underbrace{\boxed{\langle 1\rangle \varphi} \boxed{\langle 2\rangle \varphi}}_{\text{existential subformulas}}$$

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

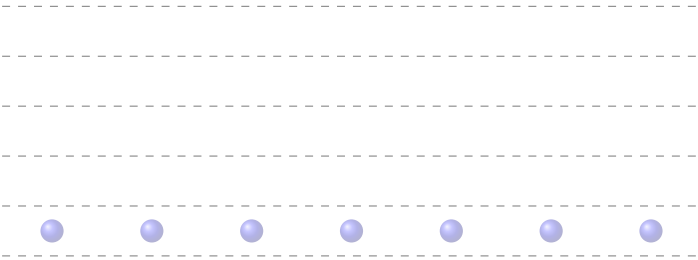
- With some additional constraints, e.g. $\neg \langle \bar{1}\rangle \top \vee \neg \langle \bar{2}\rangle \top$

Satisfiability-Testing Algorithm: Principles

Bottom-up construction of proof tree

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

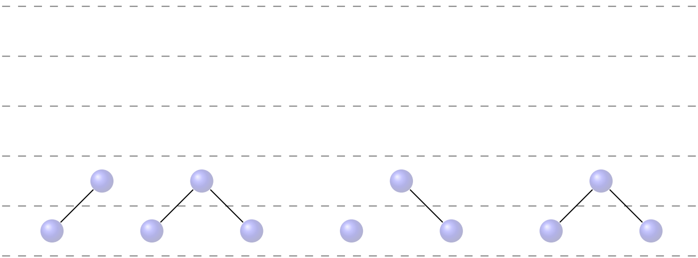
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

- Step 1: all possible leaves are added

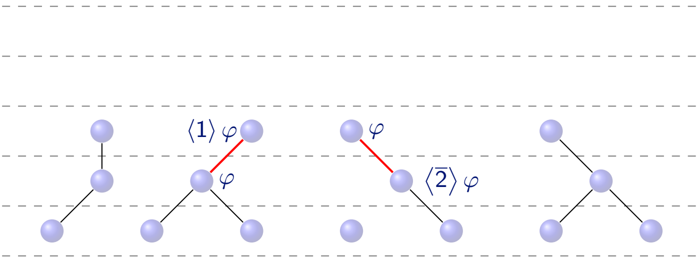
Satisfiability-Testing Algorithm: Principles



Bottom-up construction of proof tree

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

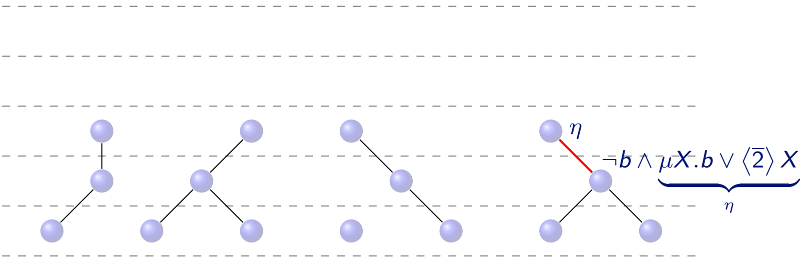
Satisfiability-Testing Algorithm: Principles



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

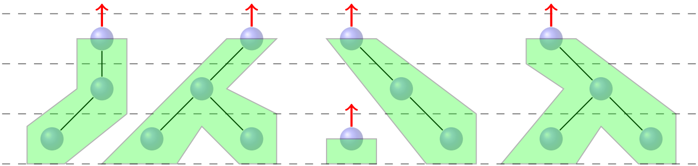
Satisfiability-Testing Algorithm: Principles



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

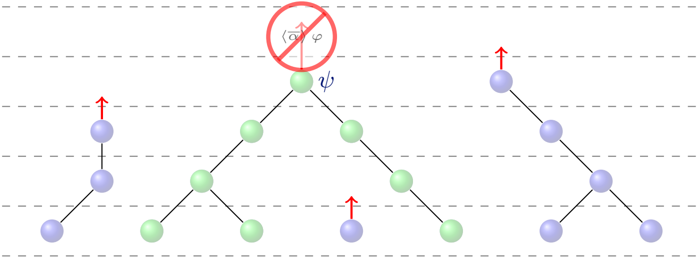
Satisfiability-Testing Algorithm: Principles



Progressive bottom-up reasoning (partial satisfiability)

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

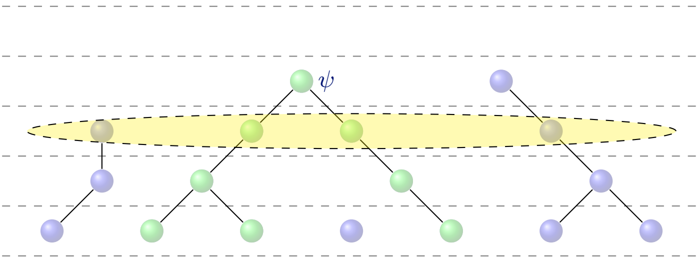
Satisfiability-Testing Algorithm: Principles



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* $n = |Lean(\psi)|$.

System fully implemented

- decision procedure
- compilers (XPath, DTD, XML Schema, CSS selectors, ...)

Overview of Some 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

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

	1968	1977	1981	1983	2006-2013
	WS2S	PDL$_{(tree)}$	CTL	μ-calculus	$\mathcal{L}_\mu$ **forward + backward (for finite trees)**
Expr.:	**MSO**	? (<MSO)	**FO**	MSO	**MSO**
Sat.:	**Non-elementary**	EXPTIME	**EXPTIME**	EXPTIME	$2^{O(n)}$
Impl.:	**MONA**	?	**?**	?	$\mathcal{L}_\mu$ **Solver**

On the practical side:

- except (hyperexponential) MONA, this is the only one implementation of a satisfiability solver for such an expressive logic
- It can be useful for graphs too: the sublogic without backward modalities enjoys the finite tree model property

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 $\rightarrow$ 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

Further Perspectives in Gaining Succinctness

Nominals

- A nominal p is an atomic proposition whose interpretation is a singleton, card(p)=1
- Captured! Idea of the translation into logic: "p and nowhereElse(p)"

$$p \wedge \neg\mathtt{descendant}(p)$$
$$\wedge \neg\mathtt{descendant\text{-}or\text{-}self}(\mathtt{following\text{-}sibling}(\mathtt{ancestor\text{-}or\text{-}self}(p)))$$

- a formula with constant-size footprint in the Lean

... Now, what about card(phi)=n ?

Further Perspectives: card(phi)=n

card(phi)=n

- Even if this remains regular, this is not a priori succinct
- For instance, L_{2a2b}: set of strings over $\Sigma = \{a, b, c\}$ containing at least 2 occurrences of a and at least two occurrences of b

Further Perspectives: card(phi)=n

card(phi)=n

- Even if this remains regular, this is not a priori succinct
- For instance, L_{2a2b}: set of strings over $\Sigma = \{a, b, c\}$ containing at least 2 occurrences of a and at least two occurrences of b

$$
\begin{gathered}
(a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star \mid \\
(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star \mid \\
(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star \mid \\
(a|b|c)^\star b(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star \mid \\
(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star \mid \\
(a|b|c)^\star b(a|b|c)^\star a(a|b|c)^\star a(a|b|c)^\star b(a|b|c)^\star
\end{gathered}
$$

Further Perspectives: card(phi)=n

- If we add $\cap$ to the regular expression operators:

$$((a|b|c)^{\star}a(a|b|c)^{\star}a(a|b|c)^{\star}) \cap ((a|b|c)^{\star}b(a|b|c)^{\star}b(a|b|c)^{\star})$$

- In logical terms, conjunction offers a dramatic reduction in expression size
- If we now consider the ability to describe numerical constraints on the frequency of occurrences, we get another exponential reduction in size:

$$((a|b|c)^{\star}a(a|b|c)^{\star})^{2} \cap ((a|b|c)^{\star}b(a|b|c)^{\star})^{2}$$

- Crucial when the complexity of the decision procedure depends on the formula size

Further Perspectives: card(phi)=n

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)

Facts

Nominals + Backward modalities + card(phi)=n

- undecidable over graphs [Bonatti-AI'04]
- decidable over finite trees

Ongoing research...

- What is the precise complexity for card(phi)=n for finite trees?
- ... or more generally of rich logical combinators that may duplicate formulas of arbitrary length (but in a particular manner)?

$\rightarrow$ Hint: look at the factorization power of the Lean

Further Perspectives: Follow the Arrows

- So far: logical description of structural constraints stemming from queries and schemas
- Can we also logically capture a notion of computation performed by programs (i.e. functions)?
- For example, can the logic capture the type algebra on which CDuce sits?

τ	$::=$			
		$\boldsymbol{b}$		**basic type**
	$\mid$	$\tau \times \tau$		**product type**
	$\mid$	$\tau \rightarrow \tau$		**function type**
	$\mid$	$\tau \vee \tau$		**union type**
	$\mid$	$\neg \tau$		**complement type**
	$\mid$	$\mathbf{0}$		**empty type**
	$\mid$	$\boldsymbol{v}$		**recursion variable**
	$\mid$	$\mu \boldsymbol{v} . \tau$		**recursive type**

- Yes. We interpret the type algebra in a purely logical manner...

Further Perspectives: Follow the Arrows

Representing functions

$f = \{(d_1, d'_1), (d_2, d'_2), \ldots\}$ modelizes a function such that:

- $f\ d_i$ may evaluate (nondeterministically) to d'_i
- $f\ x$ where $x \notin \{d_i\}$ never terminates (and is well-typed)
- if $d'_i = ERR$ then $f\ d_i$ is a type error

Lemma (Frisch et al.): considering only finite such sets of pairs is sufficient for defining semantic subtyping.

Further Perspectives: Follow the Arrows

Types as Logical Formulas (detailed encoding in [ICFP'11])

- Interpretation of $\tau_1 \to \tau_2$: all finite fs such that $f : \tau_1 \to \tau_2$
- $\mathsf{form}(\tau_1 \to \tau_2) = (\to) \wedge [1]\,\mu X.([2]\,X \wedge \langle 1 \rangle\,(\neg\mathsf{form}(\tau_1) \vee \langle 2 \rangle\,\mathsf{form}(\tau_2))))$
- with the shorthand $[\alpha]\,\varphi = \neg\,\langle\alpha\rangle\,\top \vee \langle\alpha\rangle\,\varphi$
- Intuitively: "a $(\to)$ node whose first child, if it exists, satisfies X"
- where X = "a node whose next sibling, if it exists, satisfies X, and which has a first child which either does not satisfy $\mathsf{form}(\tau_1)$ or has a next sibling which satisfies $\mathsf{form}(\tau_2)$."

Further Perspectives: Parametric Polymorphism

- We can go even further and support parametric polymorphism
- We add type variables α to the type algebra
- Intuition of subtyping in the presence of type variables:
  $\tau_1(\overline{\alpha}) \leqslant \tau_2(\overline{\alpha})$ whenever, independently of the variables $\overline{\alpha}$, any value of type τ_1 has type τ_2 as well.

$\rightarrow$ Neat formal definition of subtyping by Castagna and Xu (ICFP'11)

$\rightarrow$ Complete logical encoding in [ICFP'11] (Gesbert, Genevès and Layaïda)

- We can solve subtyping with the satisfiability solver

Interesting facts

- **The complexity bound is not affected:** $2^{\mathcal{O}(|\tau_1|+|\tau_2|)}$ **for checking** $\tau_1 \leq \tau_2$
- The $\mathcal{L}_\mu$ logic is expressive and robust by (intricate) extension

Further Perspectives: Type Synthesis

- **Objective:** static type checking for programming languages that do not require type annotations
- **Method:** (i) type inference, (ii) containment check (unsatisfiability check)
- If the containment check fails between the inferred type and e.g. the expected output type, an error is reported
- **Novelty:** Take advantage of the logic succinctness to represent inferred type portions (ongoing research...)
- A possible application: enhancing static type checking for XQuery
- Current XQuery standardized type system is unsound so far
    - if a program involves an upward navigation such as parent::*, the type Any (true in logic) is inferred
    - false negatives may be reported

Some Already Investigated Applications

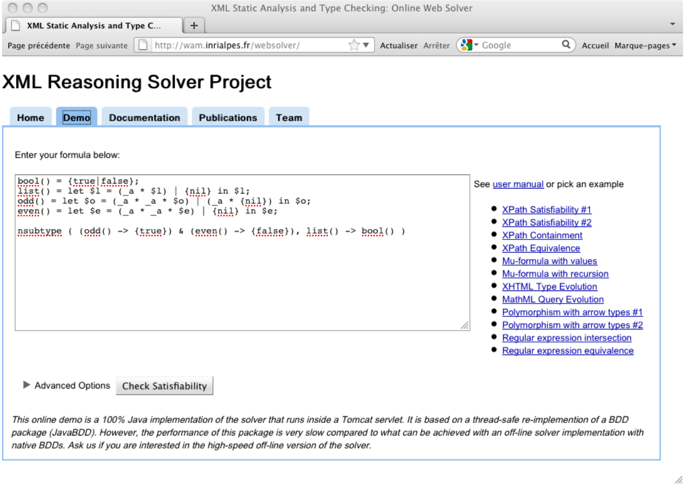
- Containment for XML queries [PLDI'07, ICDE'10]

  $\rightarrow$ equivalence test for monadic queries: $\forall t, \forall n \in t, \quad q_1(t,n) \stackrel{?}{=} q_2(t,n)$
- Modeling interleaving and counting [IJCAI'11]
- Dead code analysis for XQuery [ICSE'10, ICSE'11]
- Impact of schema evolution [ICFP'09, TOIT'11]

  $\rightarrow$ Schema S evolves into S': impact on a query written against S?
- Deciding subtyping for rich type algebras [ICFP'11]

  $\rightarrow$ Intersection, negation, function, and polymorphic types
- Containment for SPARQL queries (polyadic, graphs) under constraints [AAAI'12, IJCAR'12]
- CSS Analysis [WWW'12]

Try it online*: http://wam.inrialpes.fr/websolver



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

Long-Term Goal

Long-term view

Heterogeneity is here to stay: JSON (JS serialization) + XML + RDF (knowledge)

A unified verification toolbox

- for type-checking web programs: XQuery, XPath, "X...", Jaql etc.
- for reasoning at the layout level: CSS
- for supporting heterogenous and rich data values: XML, RDF, JSON ...
- possibly constrained by some schema languages (XML Schema, RDFS, Schematron, etc.)

Long-Term Goal

Long-term view

Heterogeneity is here to stay: JSON (JS serialization) + XML + RDF (knowledge)

A unified verification toolbox

- for type-checking web programs: XQuery, XPath, “X...”, Jaql etc.
- for reasoning at the layout level: CSS
- for supporting heterogenous and rich data values: XML, RDF, JSON ...
- possibly constrained by some schema languages (XML Schema, RDFS, Schematron, etc.)