

Expressive Logical Combinators for Free

Pierre Genevès (CNRS)

Joint work with Alan Schmitt (Inria)

IJCAI, July 30th 2015, Buenos Aires

Query Analysis

Constructs and Languages

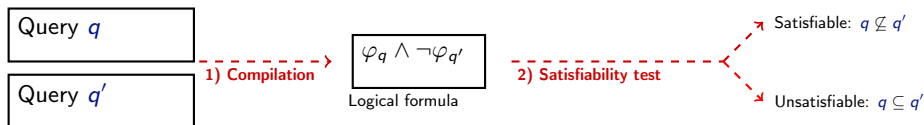
• query	q	XPath, JAQL, SPARQL
• database	d	XML, JSON, RDF
• schema	S	DTD, OWL

Problems and Applications

• Query containment	$q(d) \subseteq q'(d)$ for all d	the view update problem
• Query equivalence	$q(d) = q'(d)$ for all d	optimization of programs
• Query satisfiability	$q(d) \neq \emptyset$ for some d	dead code analysis
• Static type-checking	$q(d) \vdash S'$ for all $d \vdash S$	proving program correctness

The Logical Approach for Query Analysis

Reducing e.g. Query Containment to Logical Satisfiability:



Complexity

- Complexity of satisfiability test depends on |formula|
- Blow-ups in the logical translation increase **combined complexity**
- **Succinctness** of the logic is crucial!
 - Improving it: addressing more problems while reducing combined complexity

Example

- Set of strings over $\Sigma = \{a, b, c\}$ containing at least 2 occurrences of “a” and at least 2 occurrences of “b”:

Example

- Set of strings over $\Sigma = \{a, b, c\}$ containing at least 2 occurrences of “a” and at least 2 occurrences of “b”:

$$L_{2a2b} = \begin{aligned} & (a|b|c)^* a(a|b|c)^* a(a|b|c)^* b(a|b|c)^* b(a|b|c)^* | \\ & (a|b|c)^* a(a|b|c)^* b(a|b|c)^* a(a|b|c)^* b(a|b|c)^* | \\ & (a|b|c)^* a(a|b|c)^* b(a|b|c)^* b(a|b|c)^* a(a|b|c)^* | \\ & (a|b|c)^* b(a|b|c)^* b(a|b|c)^* a(a|b|c)^* a(a|b|c)^* | \\ & (a|b|c)^* b(a|b|c)^* a(a|b|c)^* b(a|b|c)^* a(a|b|c)^* | \\ & (a|b|c)^* b(a|b|c)^* a(a|b|c)^* a(a|b|c)^* b(a|b|c)^* \end{aligned}$$

Example

- Set of strings over $\Sigma = \{a, b, c\}$ containing at least 2 occurrences of “a” and at least 2 occurrences of “b”:

$$L_{2a2b} = \begin{aligned} & (a|b|c)^* a(a|b|c)^* a(a|b|c)^* b(a|b|c)^* b(a|b|c)^* | \\ & (a|b|c)^* a(a|b|c)^* b(a|b|c)^* a(a|b|c)^* b(a|b|c)^* | \\ & (a|b|c)^* a(a|b|c)^* b(a|b|c)^* b(a|b|c)^* a(a|b|c)^* | \\ & (a|b|c)^* b(a|b|c)^* b(a|b|c)^* a(a|b|c)^* a(a|b|c)^* | \\ & (a|b|c)^* b(a|b|c)^* a(a|b|c)^* b(a|b|c)^* a(a|b|c)^* | \\ & (a|b|c)^* b(a|b|c)^* a(a|b|c)^* a(a|b|c)^* b(a|b|c)^* \end{aligned}$$

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

$$L_{2a2b} = ((a|b|c)^* a(a|b|c)^* a(a|b|c)^*) \cap ((a|b|c)^* b(a|b|c)^* b(a|b|c)^*)$$

- \cap offers a dramatic **reduction in expression size!**
- Crucial when complexity of the decision procedure depends on |formula|
- **More generally, how can we increase succinctness?**

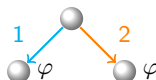
Logical Combinators

A Combinator is a Predicate that:

- takes logical formulas as **input** (and outputs a logical formula)
- might arbitrarily **duplicate** input formulas in its definition

Examples (using branching logic from [Genevès-PLDI'07])

- $\text{split}(\varphi) = \langle 1 \rangle \varphi \wedge \langle 2 \rangle \varphi$



- Order relations between tree nodes (e.g. depth-first tree traversal)

$$\text{next}(\varphi) = \underbrace{\langle 1 \rangle \mu z. \varphi \vee \langle 1 \rangle z \vee \langle 2 \rangle z}_{\text{descendant}(\varphi)} \vee \dots \underbrace{\dots}_{\text{following}(\varphi)}$$

- Regular queries with counting e.g. “at least 3 occurrences of φ ”:
 $\text{threeOrMore}(\varphi) = \text{next}(\varphi \wedge \text{next}(\varphi \wedge \text{next}(\varphi)))$

Results on Combinators

- Combinators form an **expressive** and **succinct** logical language (regular tree and path languages, counting...)
- Proof: combinators **do not increase the complexity** of decision procedures à la [Genevès-TOCL15] which stays in $2^{\mathcal{O}(|\varphi|)}$ (MSO-complete logic)

Concrete Problem	$ \varphi $	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
Analysis of CSS style sheets (IJCAI'15)	60	40 ms
Precise static type-checking for XQuery (ICFP'15)	70	35 ms

Table: Experimental Results.

Try Combinators*: <http://tyrex.inria.fr/websolver>

XML Static Analysis and Type Checking: Online Web Solver

XML Static Analysis and Type C... +

Page précédente Page suivante Actualiser Arrêter Google Accueil Marque-pages ▾

XML Reasoning Solver Project

Home **Demo** Documentation Publications Team

Enter your formula below:

```
bool() = {true|false};
list() = let $l = (_a * $l) | {nil} in $l;
odd() = let $o = (_a * _a * $o) | (_a * {nil}) in $o;
even() = let $e = (_a * _a * $e) | {nil} in $e;
nsuptype ( (odd() -> {true}) & (even() -> {false}), list() -> bool() )
```

See [user manual](#) or pick an example

- [XPath Satisfiability #1](#)
- [XPath Satisfiability #2](#)
- [XPath Containment](#)
- [XPath Equivalence](#)
- [Mu-formula with values](#)
- [Mu-formula with recursion](#)
- [XHTML Type Evolution](#)
- [MathML Query Evolution](#)
- [Polymorphism with arrow types #1](#)
- [Polymorphism with arrow types #2](#)
- [Regular expression intersection](#)
- [Regular expression equivalence](#)

▶ Advanced Options

This online demo is a 100% Java implementation of the solver that runs inside a Tomcat servlet. It is based on a thread-safe re-implementation of a BDD package (JavaBDD). However, the performance of this package is very slow compared to what can be achieved with an off-line solver implementation with native BDDs. Ask us if you are interested in the high-speed off-line version of the solver.

* and build your own predicate language for addressing your problem!