Looping Caterpillars
Searching for an algebra for XML

Loredana Afanasiev
Work by Evan Goris and Maarten Marx

Informatics Institute
University of Amsterdam
The Netherlands
Motivation

• Powerful duality in the relational world:

  Operational
  Algebra
  Query plans
  Relational algebra

  Declarative
  Logic
  Queries
  SQL

  Codd completeness

• We want the same in the XML world.
The goal

- Extra: Characterization in terms of automata and logic

Operational Algebra?

- DTD
- XPath
- Regular paths

Declarative Codd completeness

execution
Algebra

• An algebra is a set of things closed under its operations.

• Things: We choose for sets of paths (binary relations) within one XML document.

• Generators:
  – The four simplest paths in an ordered tree: child, parent, left, right
  – The node tags

• Operations: ?
Example: an algebra for Core XPath (Gottlob et al.)

- **Generators**:  
  - Basic axis: child, parent, child*, parent*, right*, left*  
  - Node tags

- **Operations**:  
  - Concatentation /  
  - Union U  
  - Counterdomain ~  
    \[(\sim R = \{(x,x) | \text{there is no } y \text{ such that } xRy\})\]
Codd-completeness of Core XPath algebra

• Theorem: (Marx, de Rijke) This algebra is Codd-complete for Core XPath.
**Evaluation**

**Problem:** This algebra is not expressive enough:
- DTD’s validation cannot be expressed;
- Not all regular path queries can be expressed.

**Tryout:**
- Add the Kleene star as an operator.
Evaluation

Operational
Core XPath
+ Kleene star

Declarative
✓ DTD
✓ XPath
✓ Regular paths

• **Bonus:** all of FO[child+, right+] is expressible.
• **But ...**
But...

• Problem:
  – Unknown expressive power both in terms of logic and in terms of automata.

• Tryout
  – Restrict Counterdomain ~
Caterpillars
(Bruggemann-Klein et al.)

- **Generators:**
  - child, parent, left, right, root, leaf, first, last;
  - Node tags;

- **Operations:**
  - Concatenation \( / \);
  - Union \( U \);
  - Reflexive, transitive closure \( * \);
Examples

self::*[child::*p] ≡ child/p/parent
Examples

\[ \text{self::*[parent::*]} \equiv \]
\[ \text{first/parent/child/first} \cup \text{left/right} \]
Caterpillars lose their way

\[ \text{self:::*[parent:::*p]} \equiv \text{parent/p/child?} \]

**Fact:** \( \text{self:::*[parent:::*p]} \) cannot be expressed by a caterpillar expression.
Caterpillars are nice

- DTD validation can be expressed by a caterpillar expression (Bruggemann-Klein et al.).
- Caterpillars are characterized in terms of tree walk automata.
Looping caterpillars

• Caterpillars plus an extra operation: $(\cdot)^{\text{loop}}$
  \[ xR^{\text{loop}} y \text{ if and only if } xRy \text{ and } x=y \]

• Now we can express
  \[
  \text{self::*[parent::p]} = (\text{parent/p/child})^{\text{loop}}
  \]
Evaluation

Operational
Looping Caterpillars!

Declarative
✓ DTD
✓ XPath
✓ Regular paths

Codd completeness

• Extra:
  – all of FO[child+,right+] is expressible.
  – Characterized by tree walk automata that can return to a point: non-inspecting pebble tree walk automata.
Conclusions and future

• Looping Caterpillars are a natural benchmark in the development of XML query languages.
• Query evaluation complexity: polynomial in combined size

• Many questions remain open though:
  – Expressive power in terms of logic (MSO or less).
  – How to implement it?
Thank you!