

XPath from a Logical Point of View

Tadeusz Litak
(joint work with
Balder ten Cate, Maarten Marx and Gaëlle Fontaine)

Department of Computer Science
University of Leicester

15 December 2010
MGS Christmas Dinner Apéritif

Advertisement

Expressive power:

Marx and De Rijke. *Semantic characterizations of navigational XPath*. SIGMOD Record 34(2), 2005

Ten Cate, Fontaine and Litak. *Some modal aspects of XPath*. M4M'07. Journal version to appear in a special "20th birthday" issue of JANCL

Ten Cate and Segoufin. *XPath, transitive closure logic, and nested tree walking automata*. Journal of the ACM, 2010. Extended abstract appeared in PODS 2008.

Place/Segoufin in LICS 2010, Fontaine/Place in MFCS 2010 ...

Axiomatization:

Ten Cate, Fontaine and Litak. *Some modal aspects of XPath*. M4M'07. Journal version to appear in a special "20th birthday" issue of JANCL

Ten Cate, Litak and Marx. Complete axiomatizations of XPath fragments. JAL 2010. Extended abstract presented at LiD 2008.

Ten Cate and Marx. *Axiomatizing the logical core of XPath 2.0*. ICDT'07.

Complexity:

Gottlob, Koch and Pichler. *Efficient algorithms for processing XPath queries*. TODS 30(2), 2005

Ten Cate and Lutz. *Query containment in very expressive XPath dialects*. PODS'07.

What This Talk Is About

Abstract of Abstract

I will sketch how to use an intimate relationship between

- subsets and extensions of XPath 1.0 and 2.0 (in particular for their "navigational core") and
- well-understood logical and algebraic formalisms

to derive results on

- complete axiomatizations
- computational complexity and
- expressive power

for XPath fragments

XML and Semi-structured Data

XML and Semi-structured Data

XML

eXtensible Markup Language

XML

eXtensible Markup Language

- began as a subset of SGML:

Standard Generalized Markup Language

(HTML: simplified and corrupted subset of SGML)

XML

eXtensible Markup Language

- began as a subset of SGML:

Standard Generalized Markup Language

(HTML: simplified and corrupted subset of SGML)

- developed to
 - RSS

XML

eXtensible Markup Language

- began as a subset of SGML:

Standard Generalized Markup Language

(HTML: simplified and corrupted subset of SGML)

- developed to
 - RSS
 - Atom

XML

eXtensible Markup Language

- began as a subset of SGML:

Standard Generalized Markup Language

(HTML: simplified and corrupted subset of SGML)

- developed to
 - RSS
 - Atom
 - SOAP

XML

eXtensible Markup Language

- began as a subset of **SGML**:

Standard Generalized Markup Language

(HTML: simplified and corrupted subset of SGML)

- developed to
 - **RSS**
 - **Atom**
 - **SOAP**
 - **XHTML ...**

Example Document

No XML talk can do without its own example document:

Example Document

No XML talk can do without its own example document:

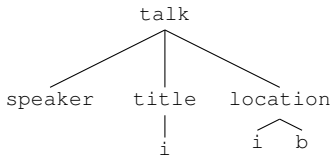
```
<?xml version='1.0' encoding='UTF-8' ?>
<talk date='15-Dec-2010'>
  <speaker uni='Leicester'>T. Litak</speaker>
  <title>
    <i>XPath</i>from a Logical Point of View
  </title>
  <location>
    <i>ATT LT3</i><b>Leicester</b>
  </location>
</talk>
```

(no DTD given, but you can easily come up with one)

What we'll see through our dim glasses

What we'll see through our dim glasses

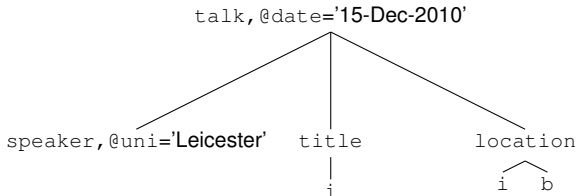
Either this ...



(we cannot even see attributes, each node is labelled with a single label: its name)

What we'll see through our dim glasses

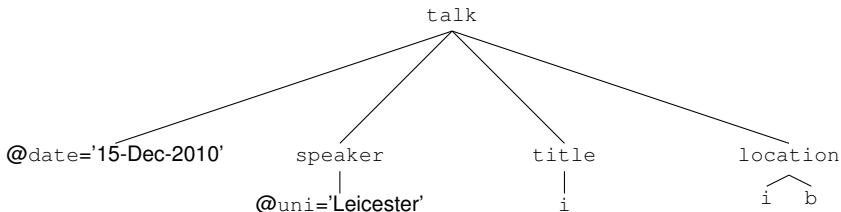
or that ...



(attribute-value pairs are additional labels)

What we'll see through our dim glasses

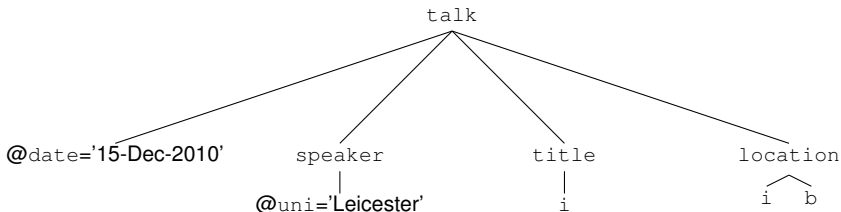
or perhaps ...



(back to the unique labelling idea, attribute-value pairs are a special kind of children)
see Ranko Lazic's talk last week for a more sophisticated approach to trees with values

What we'll see through our dim glasses

or perhaps ...



(back to the unique labelling idea, attribute-value pairs are a special kind of children)
see Ranko Lazić's talk last week for a more sophisticated approach to trees with values

At any rate, we are too blind to see actual text content

XPath 1.0: W3C Specification

- *Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer*

XPath 1.0: W3C Specification

- *Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer*
- **Primary purpose: to address parts of an XML document**

XPath 1.0: W3C Specification

- *Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer*
- **Primary purpose: to address parts of an XML document**
- ***In support of this primary purpose***, it also provides *basic facilities for manipulation of strings, numbers and booleans*

XPath 1.0: W3C Specification

- *Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer*
- **Primary purpose: to address parts of an XML document**
- ***In support of this primary purpose***, it also provides *basic facilities for manipulation of strings, numbers and booleans*
- Uses a ***compact, non-XML syntax*** to facilitate use of XPath within URIs and XML attribute values

XPath 1.0: W3C Specification

- Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer
- **Primary purpose: to address parts of an XML document**
- **In support of this primary purpose**, it also provides basic facilities for manipulation of strings, numbers and booleans
- Uses a **compact, non-XML syntax** to facilitate use of XPath within URIs and XML attribute values
- Operates on the **abstract, logical structure of an XML document**, rather than its surface syntax

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`
- **String functions.** For example:
`/note[substring(body, 1, 3) = "It' s"]`

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`
- **String functions.** For example:
`/note[substring(body, 1, 3) = "It' s"]`
- **Arithmetical functions.** ...

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`
- **String functions.** For example:
`/note[substring(body, 1, 3) = "It' s"]`
- **Arithmetical functions. ...**
- ...

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`
- **String functions.** For example:
`/note[substring(body, 1, 3) = "It' s"]`
- **Arithmetical functions.** ...
- ...
- XPath 1.0 specification (W3C, Nov '99): 37 pages.

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`
- **String functions.** For example:
`/note[substring(body, 1, 3) = "It' s"]`
- **Arithmetical functions.** ...
- ...
- XPath 1.0 specification (W3C, Nov '99): 37 pages.
- XPath 2.0 specification (W3C, Jan '07): 122 pages.

Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//i`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`
- **String functions.** For example:
`/note[substring(body, 1, 3) = "It' s"]`
- **Arithmetical functions. ...**
- ...
- XPath 1.0 specification (W3C, Nov '99): 37 pages.
- XPath 2.0 specification (W3C, Jan '07): 122 pages.
- XPath 3.0: ... ?

Michael Kay, “XPath 2.0 Programmer’s Reference”

One of the pleasures of XPath 1.0 (like XML itself) was the brevity of the specification: a mere 30 pages. I haven’t tried to print the XPath 2.0 specification, but it is certainly vastly longer. What’s more, it is split between multiple documents. The main language specification at <http://www.w3.org/TR/xpath20> points to subsidiary documents describing the data model, the function library, and the formal semantics, all at considerable length. **As with a comparison between the US Constitution and the proposed EU Constitution, the length of the document tells us more about the number of people involved in defining it than about the benefits it offers.** I would estimate that in reality the 2.0 language is about twice the size of XPath 1.0. (...) But **whether the increased word count in the spec adds precision and clarity, or merely creates opportunities for errors and inconsistencies to creep in, is anyone’s guess.**

We focus on **the basic navigational functionality of XPath:**

We focus on **the basic navigational functionality of XPath:**
(no arithmetics, no strings, no counting . . .
—recall these features are secondary!)

We focus on **the basic navigational functionality of XPath**:
(no arithmetics, no strings, no counting . . .
—recall these features are secondary!)

Core XPath 1.0

- Isolated by Gottlob, Koch and Pichler in 2002
- Explicit motivation: allows **linear time model checking** on XML trees
- An additional advantage of such a simple language: data model discrepancies between XPath 1.0 and 2.0 no longer relevant

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

$s ::= \Downarrow, \Uparrow, \Leftarrow, \Rightarrow$

$a ::= s \mid s^+$

$pexpr ::= a \mid \cdot \mid pexpr/pexpr \mid pexpr \cup pexpr \mid pexpr[nexpr]$

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

$s ::= \Downarrow, \Uparrow, \Leftarrow, \Rightarrow$

$a ::= s \mid s^+$

$pexpr ::= a \mid \cdot \mid pexpr/pexpr \mid pexpr \cup pexpr \mid pexpr[nexpr]$

$nexpr ::= p \mid \langle pexpr \rangle \mid \neg nexpr \mid nexpr \vee nexpr \quad (p \in \Sigma)$

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

```
children::*,parent::*,preceding-sibling::*[1],following-sibling::*[1]
a ::= s | s+
pexpr ::= a | · | pexpr/pexpr | pexpr ∪ pexpr | pexpr[nexpr]
nexpr ::= p | ⟨pexpr⟩ | ¬nexpr | nexpr ∨ nexpr (p ∈ Σ)
```

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

```
children::*,parent::*,preceding-sibling::*[1],following-sibling::*[1]
...descendant::*,ancestor::*,preceding-sibling::*[1],following-sibling::*[1]
pexpr ::= a | · | pexpr/pexpr | pexpr ∪ pexpr | pexpr[nexpr]
nexpr ::= p | ⟨pexpr⟩ | ¬nexpr | nexpr ∨ nexpr (p ∈ Σ)
```

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

```
children::*,parent::*,preceding-sibling::*[1],following-sibling::*[1]
...descendant::*,ancestor::*,preceding-sibling::*,following-sibling::*
...self::*, pexpr/pexpr, pexpr | pexpr, pexpr[nexpr]
nexpr ::= p | (pexpr) | ¬nexpr | nexpr ∨ nexpr    (p ∈ Σ)
```

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

```
children::*,parent::*,preceding-sibling::*[1],following-sibling::*[1]
...descendant::*,ancestor::*,preceding-sibling::*[1],following-sibling::*[1]
...self::*, pexpr/pexpr, pexpr | pexpr, pexpr[nexpr]
self::p, pexpr, not(nexpr), nexpr or nexpr
```

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

$s ::= \Downarrow, \Uparrow, \Leftarrow, \Rightarrow$

$a ::= s \mid s^+$

$pexpr ::= a \mid \cdot \mid pexpr/pexpr \mid pexpr \cup pexpr \mid pexpr[nexpr]$

$nexpr ::= p \mid \langle pexpr \rangle \mid \neg nexpr \mid nexpr \vee nexpr \quad (p \in \Sigma)$

(Our notation is a *bit* different from the official XPath notation)

Core XPath has two types of expressions:

- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

$s ::= \Downarrow, \Uparrow, \Leftarrow, \Rightarrow$

$a ::= s \mid s^+$

$pexpr ::= a \mid \cdot \mid pexpr/pexpr \mid pexpr \cup pexpr \mid pexpr[nexpr]$

$nexpr ::= p \mid \langle pexpr \rangle \mid \neg nexpr \mid nexpr \vee nexpr \quad (p \in \Sigma)$

(Our notation is a *bit* different from the official XPath notation)

We also consider **single and restricted axis fragments** of CoreXPath—notation **CoreXPath(*a*)** for a single axis
and **CoreXPath(*A*)** for a set of axes

Restricting the set of axes is quite common: recall James Cheney's or Ranko Lazić's talks last week for fresh examples

Semantics of CoreXPath

As said above, we see XML documents as
finite sibling-ordered node labelled trees:
ideal abstraction for such a simple syntax

As said above, we see XML documents as
finite sibling-ordered node labelled trees:
ideal abstraction for such a simple syntax

XML document

A tuple $T = (N, R_{\downarrow}, R_{\Rightarrow}, V)$ where

As said above, we see XML documents as
finite sibling-ordered node labelled trees:
ideal abstraction for such a simple syntax

XML document

A tuple $T = (N, R_{\downarrow}, R_{\Rightarrow}, V)$ where

- N is the set of nodes,

As said above, we see XML documents as
finite sibling-ordered node labelled trees:
ideal abstraction for such a simple syntax

XML document

A tuple $T = (N, R_{\Downarrow}, R_{\Rightarrow}, V)$ where

- N is the set of nodes,
- R_{\Downarrow} and R_{\Rightarrow} are 'child' and 'next sibling' relations of a finite tree, and

As said above, we see XML documents as
finite sibling-ordered node labelled trees:
ideal abstraction for such a simple syntax

XML document

A tuple $T = (N, R_{\Downarrow}, R_{\Rightarrow}, V)$ where

- N is the set of nodes,
- R_{\Downarrow} and R_{\Rightarrow} are ‘child’ and ‘next sibling’ relations of a finite tree, and
- $V : \Sigma \rightarrow 2^N$ (or just $V : N \rightarrow \Sigma$ if unique labelling assumed)

Semantics of Core XPath (ct'd)

pexpr : pairs (context node, reachable node)—subsets of N^2

$$\llbracket s \rrbracket^T = R_s$$

$$\llbracket s^+ \rrbracket^T = \text{the transitive closure of } R_s$$

$$\llbracket \cdot \rrbracket^T = \text{the identity relation on } N$$

$$\llbracket A/B \rrbracket^T = \text{composition of } \llbracket A \rrbracket^T \text{ and } \llbracket B \rrbracket^T$$

$$\llbracket A \cup B \rrbracket^T = \text{union of } \llbracket A \rrbracket^T \text{ and } \llbracket B \rrbracket^T$$

$$\llbracket A[\phi] \rrbracket^T = \{(n, m) \in \llbracket A \rrbracket^T \mid m \in \llbracket \phi \rrbracket^T\}$$

nexpr : subsets of N

$$\llbracket p \rrbracket^T = \{n \in N \mid n \in V(p)\}$$

$$\llbracket \phi \wedge \psi \rrbracket^T = \llbracket \phi \rrbracket^T \cap \llbracket \psi \rrbracket^T$$

$$\llbracket \neg \phi \rrbracket^T = N \setminus \llbracket \phi \rrbracket^T$$

$$\llbracket \langle A \rangle \rrbracket^T = \text{domain of } \llbracket A \rrbracket^T = \{n \mid (n, m) \in \llbracket A \rrbracket^T\}$$

A (slightly modified) diagram of Johan Van Benthem

W

unary properties

→

modes

→

of states

←

projections

←

propositional operators

ML

W²

binary relations

between states

program operators

DRA/TRA

Examples of modes:

$$?X := \{\langle x, x \rangle \mid x \in X\} \quad (\text{testing})$$

$$!X := \{\langle w, x \rangle \mid w \in \underline{\mathfrak{W}}, x \in X\} \quad (\text{realizing})$$

Examples of projections:

$$\langle R \rangle := \{w \in \underline{\mathfrak{W}} \mid \exists v \in \underline{\mathfrak{W}}. wR^{\mathfrak{W}}v\} \quad (\text{domain})$$

$$\pi^{-1}(R) := \{w \in \underline{\mathfrak{W}} \mid \exists v \in \underline{\mathfrak{W}}. vR^{\mathfrak{W}}w\} \quad (\text{codomain})$$

$$\sim R := \{w \in \underline{\mathfrak{W}} \mid \forall v \in \underline{\mathfrak{W}}. \neg(wR^{\mathfrak{W}}v)\} \quad (\text{antidomain})$$

$$\Delta(R) := \{w \in \underline{\mathfrak{W}} \mid wR^{\mathfrak{W}}w\} \quad (\text{diagonal})$$

Examples of modes:

$$?X := \{\langle x, x \rangle \mid x \in X\} \quad \text{(testing)}$$

$$!X := \{\langle w, x \rangle \mid w \in \underline{\mathbb{W}}, x \in X\} \quad \text{(realizing)}$$

Examples of projections:

$$\langle R \rangle := \{w \in \underline{\mathbb{W}} \mid \exists v \in \underline{\mathbb{W}}. wR^{\mathbb{W}}v\} \quad \text{(domain)}$$

$$\pi^{-1}(R) := \{w \in \underline{\mathbb{W}} \mid \exists v \in \underline{\mathbb{W}}. vR^{\mathbb{W}}w\} \quad \text{(codomain)}$$

$$\sim R := \{w \in \underline{\mathbb{W}} \mid \forall v \in \underline{\mathbb{W}}. \neg(wR^{\mathbb{W}}v)\} \quad \text{(antidomain)}$$

$$\Delta(R) := \{w \in \underline{\mathbb{W}} \mid wR^{\mathbb{W}}w\} \quad \text{(diagonal)}$$

NOTE THAT:

$$\begin{aligned} \langle R \rangle &= \sim\sim R \\ &= R/R^{\sim} \cap \cdot \end{aligned}$$

$$\Delta(R) = R \cap \cdot$$

$$\pi^{-1}(R) = \langle R^{\sim} \rangle$$

Comments for logicians

- Note we do **not allow transitive closure of arbitrary path expressions** (allowed in non-standard extensions like Regular XPath)
- Note also that path expressions, as opposed to node expressions, are **not closed under other boolean connectives** than sum (changed in XPath 2.0)

Comments for logicians

- Note we do **not allow transitive closure of arbitrary path expressions** (allowed in non-standard extensions like Regular XPath)
- Note also that path expressions, as opposed to node expressions, are **not closed under other boolean connectives** than sum (changed in XPath 2.0)
- Therefore, we are **not exactly in the world of Tarski's relation algebras**

Comments for logicians

- Note we do **not allow transitive closure of arbitrary path expressions** (allowed in non-standard extensions like Regular XPath)
- Note also that path expressions, as opposed to node expressions, are **not closed under other boolean connectives** than sum (changed in XPath 2.0)
- Therefore, we are **not exactly in the world of Tarski's relation algebras**
- The right algebraic two-sorted setting would be **boolean modules over idempotent semirings**

Comments for logicians

- Note we do **not allow transitive closure of arbitrary path expressions** (allowed in non-standard extensions like Regular XPath)
- Note also that path expressions, as opposed to node expressions, are **not closed under other boolean connectives** than sum (changed in XPath 2.0)
- Therefore, we are **not exactly in the world of Tarski's relation algebras**
- The right algebraic two-sorted setting would be **boolean modules over idempotent semirings**
- It is possible to move the discussion to one-sorted setting, though:

Dynamic Relation Algebras (DRA's)
studied in the 1990's by a Dutch group in Utrecht (A. Visser, M. Hollenberg)

Comments for logicians

- Note we do **not allow transitive closure of arbitrary path expressions** (allowed in non-standard extensions like Regular XPath)
- Note also that path expressions, as opposed to node expressions, are **not closed under other boolean connectives** than sum (changed in XPath 2.0)
- Therefore, we are **not exactly in the world of Tarski's relation algebras**
- The right algebraic two-sorted setting would be **boolean modules over idempotent semirings**
- It is possible to move the discussion to one-sorted setting, though:

Dynamic Relation Algebras (DRA's)

studied in the 1990's by a Dutch group in Utrecht (A. Visser, M. Hollenberg)
that is, **idempotent semirings with antidomain operation \sim**

The Utrecht group used the name **dynamic negation**

Antidomain: term introduced recently by Desharnais, Jipsen and Struth

Short Core XPath

Enter the **Short CoreXPath (SCX)** of de Rijke and Marx:
one-sorted notational variant

Enter the **Short CoreXPath (SCX)** of de Rijke and Marx:
one-sorted notational variant

Syntax of Short Core XPath:

$s ::= \Downarrow, \Uparrow, \Leftarrow, \Rightarrow$

$a ::= s \mid s^+$

$\text{exp} ::= \cdot \mid a \mid \text{exp}/\text{exp} \mid \text{exp} \cup \text{exp}$

Enter the **Short CoreXPath (SCX)** of de Rijke and Marx:
one-sorted notational variant

Syntax of Short Core XPath:

$s ::= \Downarrow, \Uparrow, \Leftarrow, \Rightarrow$

$a ::= s \mid s^+$

$\text{exp} ::= \cdot \mid a \mid \text{exp}/\text{exp} \mid \text{exp} \cup \text{exp} \mid ?p \mid \sim\text{exp} \quad (p \in \Sigma)$

Enter the **Short CoreXPath (SCX)** of de Rijke and Marx:
one-sorted notational variant

Syntax of Short Core XPath:

$s ::= \Downarrow, \Uparrow, \Leftarrow, \Rightarrow$

$a ::= s \mid s^+$

$\text{exp} ::= \cdot \mid a \mid \text{exp}/\text{exp} \mid \text{exp} \cup \text{exp} \mid ?p \mid \sim\text{exp} \quad (p \in \Sigma)$

Definition of **a single/restricted axis fragment** remains the same

Semantics of Short Core XPath

pexpr : pairs (context node, reachable node)—subsets of N^2

$$\llbracket s \rrbracket^T = R_s$$

$$\llbracket s^+ \rrbracket^T = \text{the transitive closure of } R_s$$

$$\llbracket \cdot \rrbracket^T = \text{the identity relation on } N$$

$$\llbracket A/B \rrbracket^T = \text{composition of } \llbracket A \rrbracket^T \text{ and } \llbracket B \rrbracket^T$$

$$\llbracket A \cup B \rrbracket^T = \text{union of } \llbracket A \rrbracket^T \text{ and } \llbracket B \rrbracket^T$$

$$\llbracket A[\phi] \rrbracket^T = \{(n, m) \in \llbracket A \rrbracket^T \mid m \in \llbracket \phi \rrbracket^T\}$$

nexpr : subsets of N

$$\llbracket p \rrbracket^T = \{n \in N \mid n \in V(p)\}$$

$$\llbracket \phi \wedge \psi \rrbracket^T = \llbracket \phi \rrbracket^T \cap \llbracket \psi \rrbracket^T$$

$$\llbracket \neg \phi \rrbracket^T = N \setminus \llbracket \phi \rrbracket^T$$

$$\llbracket \langle A \rangle \rrbracket^T = \text{domain of } \llbracket A \rrbracket^T = \{n \mid (n, m) \in \llbracket A \rrbracket^T\}$$

Semantics of Short Core XPath

exp : pairs (context node, reachable node)—subsets of N^2

$$\llbracket s \rrbracket^T = R_s$$

$$\llbracket s^+ \rrbracket^T = \text{the transitive closure of } R_s$$

$$\llbracket \cdot \rrbracket^T = \text{the identity relation on } N$$

$$\llbracket A/B \rrbracket^T = \text{composition of } \llbracket A \rrbracket^T \text{ and } \llbracket B \rrbracket^T$$

$$\llbracket A \cup B \rrbracket^T = \text{union of } \llbracket A \rrbracket^T \text{ and } \llbracket B \rrbracket^T$$

$$\llbracket ?p \rrbracket^T = \{(n, n) \in N^2 \mid n \in V(p)\}$$

$$\llbracket \sim A \rrbracket^T = \{(n, n) \in N^2 \mid \forall m. (n, m) \notin \llbracket A \rrbracket^T\}$$

Back-and-forth Between Core XPath and SCX

One direction is easy:

$$\llbracket \sim A \rrbracket^T = \llbracket \cdot [\neg \langle A \rangle] \rrbracket^T$$

Back-and-forth Between Core XPath and SCX

One direction is easy:

$$\llbracket \sim A \rrbracket^T = \llbracket \cdot [\neg \langle A \rangle] \rrbracket^T$$

But there is also a polynomial translation t in the reverse direction:

$$t(p) = ?p$$

$$t(\langle A \rangle) = \sim \sim t(A)$$

$$t(\phi \wedge \psi) = \sim (\sim t(\phi) \cup \sim t(\psi))$$

$$t(A[\phi]) = t(A)/t(\phi)$$

other connectives being straightforward. Clearly

Back-and-forth Between Core XPath and SCX

One direction is easy:

$$\llbracket \sim A \rrbracket^T = \llbracket \cdot [\neg \langle A \rangle] \rrbracket^T$$

But there is also a polynomial translation t in the reverse direction:

$$\begin{aligned} t(p) &= ?p \\ t(\langle A \rangle) &= \sim \sim t(A) \\ t(\phi \wedge \psi) &= \sim (\sim t(\phi) \cup \sim t(\psi)) \\ t(A[\phi]) &= t(A)/t(\phi) \end{aligned}$$

other connectives being straightforward. Clearly

$$\begin{aligned} \llbracket A \rrbracket^T &= \llbracket t(A) \rrbracket^T && \text{for all } A \in \text{pexpr} \\ \llbracket \cdot [\phi] \rrbracket^T &= \llbracket t(\phi) \rrbracket^T && \text{for all } \phi \in \text{nexpr} \end{aligned}$$

When Two Queries Are Equivalent?

Definition

Let P and Q be either

- both path expressions or
- both node expressions

We say P and Q are **equivalent** ($P \equiv Q$)
if for any document $\llbracket P \rrbracket^T = \llbracket Q \rrbracket^T$

Problems Equivalent to Equivalence

- (because of presence of \forall and \cup):
 - containment for node expressions
 - containment for path expressions

Problems Equivalent to Equivalence

- (because of presence of \vee and \cup):
 - containment for node expressions
 - containment for path expressions
- (because of presence of \wedge and \neg):
 - satisfiability for node expressions

Problems Equivalent to Equivalence

- (because of presence of \vee and \cup):
 - containment for node expressions
 - containment for path expressions
- (because of presence of \wedge and \neg):
 - satisfiability for node expressions
- satisfiability for **path** expressions . . .

Problems Equivalent to Equivalence

- (because of presence of \vee and \cup):
 - containment for node expressions
 - containment for path expressions
- (because of presence of \wedge and \neg):
 - satisfiability for node expressions
- satisfiability for **path** expressions ...
... this is a bit nontrivial

Problems Equivalent to Equivalence

- (because of presence of \vee and \cup):
 - containment for node expressions
 - containment for path expressions
- (because of presence of \wedge and \neg):
 - satisfiability for node expressions
- satisfiability for **path** expressions ...
... this is a bit nontrivial
 - reduction of satisfaction to equivalence:
straightforward

Problems Equivalent to Equivalence

- (because of presence of \vee and \cup):
 - containment for node expressions
 - containment for path expressions
- (because of presence of \wedge and \neg):
 - satisfiability for node expressions
- satisfiability for **path** expressions ...
... this is a bit nontrivial
 - reduction of satisfaction to equivalence:
straightforward
 - reduction of equivalence to satisfaction:
requires *The Nasty Trick*

Which expressions are equivalent?

Let's give it a try:

Which expressions are equivalent?

Let's give it a try:

- is it true that
 $\cdot \equiv \uparrow/\downarrow?$

Which expressions are equivalent?

Let's give it a try:

- is it true that

$$\cdot \equiv \uparrow/\downarrow? \quad \text{⋮} \quad \text{↪}$$

Which expressions are equivalent?

Let's give it a try:

- is it true that

$$\cdot \equiv \uparrow/\downarrow? \quad \cdot \curvearrowright$$

- fine, how about

$$\cdot \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \cdot \cup \Rightarrow^+?$$

Which expressions are equivalent?

Let's give it a try:

- is it true that

$$\cdot \equiv \uparrow/\downarrow? \quad \text{⋮}$$

- fine, how about

$$\cdot \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \cdot \cup \Rightarrow^+? \quad \text{⋮}$$

Which expressions are equivalent?

Let's give it a try:

- is it true that

$$\cdot \equiv \uparrow/\downarrow? \quad \text{⋮}$$

- fine, how about

$$\cdot \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \cdot \cup \Rightarrow^+? \quad \text{⋮}$$

- One last try: how about

$$\cdot [\langle \downarrow \rangle] \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \cdot [\langle \uparrow \rangle] \cup \Rightarrow^+?$$

Which expressions are equivalent?

Let's give it a try:

- is it true that

$$\cdot \equiv \uparrow/\downarrow? \quad \text{⋮}$$

- fine, how about

$$\cdot \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \cdot \cup \Rightarrow^+? \quad \text{⋮}$$

- One last try: how about

$$\sim\sim\downarrow \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \sim\sim\uparrow \cup \Rightarrow^+?$$

Which expressions are equivalent?

Let's give it a try:

- is it true that

$$\cdot \equiv \uparrow/\downarrow? \quad \text{:}$$

- fine, how about

$$\cdot \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \cdot \cup \Rightarrow^+? \quad \text{:}$$

- One last try: how about

$$\sim\sim\downarrow \equiv \downarrow/\uparrow$$

and

$$\uparrow/\downarrow \equiv \leftarrow^+ \cup \sim\sim\uparrow \cup \Rightarrow^+? \quad \text{:}$$

A non-trivial problem for **query rewrite and optimization**:

Evaluation times of two equivalent queries
may differ up to **several orders of magnitude!**

A non-trivial problem for **query rewrite and optimization**:

Evaluation times of two equivalent queries
may differ up to **several orders of magnitude!**

When implementing an optimizer,
you may need thousands of those equivalences
Now how do you know. . .

A non-trivial problem for **query rewrite and optimization**:

Evaluation times of two equivalent queries
may differ up to **several orders of magnitude!**

When implementing an optimizer,
you may need thousands of those equivalences

Now how do you know. . .

(soundness problem)

. . . **all of your equivalences are valid?**

some fake equivalences not so easy to spot, especially in hurry

A non-trivial problem for **query rewrite and optimization**:

Evaluation times of two equivalent queries
may differ up to **several orders of magnitude!**

When implementing an optimizer,
you may need thousands of those equivalences
Now how do you know. . .

(soundness problem)

. . . **all of your equivalences are valid?**

some fake equivalences not so easy to spot, especially in hurry

(completeness problem)

. . . **you took care of all (possibly) relevant ones?**

there might be classes of equivalences you never thought of!

Definition (Complete Axiomatization)

A complete axiomatization of a given XPath fragment:

A set of

- finitely many valid **equivalence schemes**
- finitely many validity preserving **inference rules**

from which every other valid equivalence is derivable.

Definition (Complete Axiomatization)

A complete axiomatization of a given XPath fragment:

A set of

- finitely many valid **equivalence schemes**
- finitely many validity preserving **inference rules**

from which every other valid equivalence is derivable.

For logicians again: of course, we are interested only in finite axiomatizations. As intended models are finite, finite axiomatization implies **decidability!**

Definition (Complete Axiomatization)

A complete axiomatization of a given XPath fragment:

A set of

- finitely many valid **equivalence schemes**
- finitely many validity preserving **inference rules**

from which every other valid equivalence is derivable.

For logicians again: of course, we are interested only in finite axiomatizations. As intended models are finite, finite axiomatization implies **decidability!**

One of reasons why we consider Core XPath only:

the whole XPath would be too big to allow an axiomatization

Logic—Algebra—Query Languages

Logicians and algebraists have long studied similar problems in a different disguise:

logic:	algebras:	databases:
--------	-----------	------------

Logic—Algebra—Query Languages

Logicians and algebraists have long studied similar problems in a different disguise:

logic:	algebras:	databases:
formulas	terms	query plans

Logic—Algebra—Query Languages

Logicians and algebraists have long studied similar problems in a different disguise:

logic:	algebras:	databases:
formulas	terms	query plans
tautologies	equations	query equivalences

Logic—Algebra—Query Languages

Logicians and algebraists have long studied similar problems in a different disguise:

logic:	algebras:	databases:
formulas	terms	query plans
tautologies	equations	query equivalences
inference rules		rewrite rules

Logic—Algebra—Query Languages

Logicians and algebraists have long studied similar problems in a different disguise:

logic:	algebras:	databases:
formulas	terms	query plans
tautologies	equations	query equivalences
inference rules		rewrite rules

In particular, they standardized a beautifully simple set of validity preserving rules:

Logic—Algebra—Query Languages

Logicians and algebraists have long studied similar problems in a different disguise:

logic:	algebras:	databases:
formulas	terms	query plans
tautologies	equations	query equivalences
inference rules		rewrite rules

In particular, they standardized a beautifully simple set of validity preserving rules:

Birkhoff's Calculus For Equational Logic

Birkhoff's Calculus For Equational Logic

Definition

- Let Γ be a set of equivalences.
Equivalence $P \equiv Q$ is **derivable** from Γ if it can be obtained by the following rules:

$$P \equiv P$$

Birkhoff's Calculus For Equational Logic

Definition

- Let Γ be a set of equivalences.
Equivalence $P \equiv Q$ is **derivable** from Γ if it can be obtained by the following rules:

$$P \equiv P$$

$$P \equiv Q$$

$$\implies$$

$$Q \equiv P$$

Birkhoff's Calculus For Equational Logic

Definition

- Let Γ be a set of equivalences.

Equivalence $P \equiv Q$ is **derivable** from Γ if it can be obtained by the following rules:

$$P \equiv P$$

$$P \equiv Q \quad \Longrightarrow \quad Q \equiv P$$

$$P \equiv Q \ \& \ Q \equiv R \quad \Longrightarrow \quad P \equiv R$$

Birkhoff's Calculus For Equational Logic

Definition

- Let Γ be a set of equivalences.

Equivalence $P \equiv Q$ is **derivable** from Γ if it can be obtained by the following rules:

$$P \equiv P$$

$$P \equiv Q \quad \Longrightarrow \quad Q \equiv P$$

$$P \equiv Q \ \& \ Q \equiv R \quad \Longrightarrow \quad P \equiv R$$

$$P \equiv Q \quad \Longrightarrow \quad R \equiv R'$$

(R' is obtained from R by replacing occurrences of P by Q)

Birkhoff's Calculus For Equational Logic

Definition

- Let Γ be a set of equivalences.

Equivalence $P \equiv Q$ is **derivable** from Γ if it can be obtained by the following rules:

$$P \equiv P$$

$$P \equiv Q \quad \Longrightarrow \quad Q \equiv P$$

$$P \equiv Q \ \& \ Q \equiv R \quad \Longrightarrow \quad P \equiv R$$

$$P \equiv Q \quad \Longrightarrow \quad R \equiv R'$$

(R' is obtained from R by replacing occurrences of P by Q)

- An axiomatization using Birkhoff's rules only is **orthodox**.

Birkhoff's Calculus For Equational Logic

Definition

- Let Γ be a set of equivalences.

Equivalence $P \equiv Q$ is **derivable** from Γ if it can be obtained by the following rules:

$$P \equiv P$$

$$P \equiv Q \quad \Longrightarrow \quad Q \equiv P$$

$$P \equiv Q \ \& \ Q \equiv R \quad \Longrightarrow \quad P \equiv R$$

$$P \equiv Q \quad \Longrightarrow \quad R \equiv R'$$

(R' is obtained from R by replacing occurrences of P by Q)

- An axiomatization using Birkhoff's rules only is **orthodox**.

Clearly, these rules preserve validity.

Q1: *Why Birkhoff Calculus?*

Before we proceed, you may have two questions:

Q1: *Why Birkhoff Calculus?*

Before we proceed, you may have two questions:

Definition

What is so great about this derivation system? Is it . . .

Q1: *Why Birkhoff Calculus?*

Before we proceed, you may have two questions:

Definition

What is so great about this derivation system? Is it ...


- ... the definition itself?

Q1: Why Birkhoff Calculus?

Before we proceed, you may have two questions:

Definition

What is so great about this derivation system? Is it ...


- ... the definition itself? 
Should feel straightforward and natural,
not surprising and counterintuitive

Q1: Why Birkhoff Calculus?

Before we proceed, you may have two questions:

Definition

What is so great about this derivation system? Is it ...



- ... the definition itself? 
Should feel straightforward and natural,
not surprising and counterintuitive
- ... the avalanche of results it triggered off?

Q1: Why Birkhoff Calculus?

Before we proceed, you may have two questions:

Definition

What is so great about this derivation system? Is it ...

- ... the definition itself? 
Should feel straightforward and natural,
not surprising and counterintuitive
- ... the avalanche of results it triggered off? 
Theory of varieties developed since the 1930's: semigroups and
groups, semirings, semilattices, lattices and residuated lattices,
boolean algebras, abstract relation and cylindric algebras ...

Q1 (ct'd): *But What Use Are They For Us?*

Q1 (ct'd): *But What Use Are They For Us?*

An orthodox axiomatization

≡

An elegant, self-contained equational rewrite system
(no need to break equational reasoning with intermediate lemmas)

Q1 (ct'd): But What Use Are They For Us?

An orthodox axiomatization

≡

An elegant, self-contained equational rewrite system
(no need to break equational reasoning with intermediate lemmas)

Almost all axiomatizations presented today will be orthodox
(you're going to see one exception at the end of the talk and dislike it)

Q2: Anything Special about XPath?

Question

How about complete axiomatizations for SQL-like languages?

After all, there has been nothing XML specific to what I said . . .

Q2: Anything Special about XPath?

Question

How about complete axiomatizations for SQL-like languages?

After all, there has been nothing XML specific to what I said . . .

Answer

*Even with no more than three attributes, you soon run into **unaxiomatizability results!** (©by logicians and algebraists)*

Some database theorists got into problems not knowing about it . . .

Q2: Anything Special about XPath?

Question

How about complete axiomatizations for SQL-like languages?

After all, there has been nothing XML specific to what I said . . .

Answer

*Even with no more than three attributes, you soon run into **unaxiomatizability results!** (©by logicians and algebraists)*

Some database theorists got into problems not knowing about it . . .

It does not mean you cannot find interesting axiomatizable fragments—they are rather small though

Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer

Yes.

Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer

Yes.

Long Answer

Yes, precisely because

- *we can isolate the navigational core . . .
(would not make much sense in the relational context)*

Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer

Yes.

Long Answer

Yes, precisely because

- *we can isolate the navigational core . . .
(would not make much sense in the relational context)*
- *. . . and this core is related to
well-behaved, axiomatizable formalisms:*
 - *Node expressions—to modal logic*

Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer

Yes.

Long Answer

Yes, *precisely because*

- *we can isolate the navigational core . . .*
(would not make much sense in the relational context)
- *. . . and this core is related to*
well-behaved, axiomatizable formalisms:
 - *Node expressions—to modal logic*
 - *Path expressions—to idempotent (antidomain) semirings*

Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer

Yes.

Long Answer

Yes, *precisely because*

- *we can isolate the navigational core . . .*
(would not make much sense in the relational context)
- *. . . and this core is related to*
well-behaved, axiomatizable formalisms:
 - *Node expressions—to modal logic*
 - *Path expressions—to idempotent (antidomain) semirings*
 - *The duality of path and node expressions:*
resembles (fragments of) the logic of programs (PDL)

Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer

Yes.

Long Answer

Yes, *precisely because*

- *we can isolate the navigational core . . .*
(would not make much sense in the relational context)
- *. . . and this core is related to*
well-behaved, axiomatizable formalisms:
 - *Node expressions—to modal logic*
 - *Path expressions—to idempotent (antidomain) semirings*
 - *The duality of path and node expressions:*
resembles (fragments of) the logic of programs (PDL)

Basic Axioms I: Idempotent Semirings

$$\text{ISAx1} \quad (A \cup B) \cup C \equiv A \cup (B \cup C)$$

$$\text{ISAx2} \quad A \cup B \equiv B \cup A$$

$$\text{ISAx3} \quad A \cup A \equiv A$$

$$\text{ISAx4} \quad A/(B/C) \equiv (A/B)/C$$

$$\text{ISAx5} \left\{ \begin{array}{l} \cdot/A \\ A/\cdot \end{array} \right. \equiv A$$

$$\text{ISAx5} \left\{ \begin{array}{l} \cdot/A \\ A/\cdot \end{array} \right. \equiv A$$

$$\text{ISAx6} \left\{ \begin{array}{l} A/(B \cup C) \\ (A \cup B)/C \end{array} \right. \equiv A/B \cup A/C$$

$$\text{ISAx6} \left\{ \begin{array}{l} A/(B \cup C) \\ (A \cup B)/C \end{array} \right. \equiv A/C \cup B/C$$

$$\text{ISAx7} \quad \perp \subseteq A$$

Distributive lattices, Kleene algebras, Tarski's relation algebras:
they all have **idempotent semiring** reducts.

Idempotency is the axiom ISAx3.

\perp abbreviates $\cdot [\neg \langle \cdot \rangle]$

Basic Axioms II: Predicate Axioms

$$\text{PrAx1 } A[\neg\langle B \rangle] / B \equiv \perp$$

$$\text{PrAx2 } A[\phi \vee \psi] \equiv A[\phi] \cup A[\psi]$$

$$\text{PrAx3 } (A/B)[\phi] \equiv A/B[\phi]$$

$$\text{PrAx4 } \cdot[\langle \cdot \rangle] \equiv \cdot$$

In Tarski's relation algebras and XPath 2.0,
predicates can be defined away

Note that PrAx3 would not be valid
if we allowed **unrestricted positional predicates**

Basic Axioms III: Node Axioms

$$\text{NdAx1 } \phi \quad \equiv \quad \neg(\neg\phi \vee \psi) \vee \neg(\neg\phi \vee \neg\psi)$$

$$\text{NdAx2 } \langle A \cup B \rangle \quad \equiv \quad \langle A \rangle \vee \langle B \rangle$$

$$\text{NdAx3 } \langle A/B \rangle \quad \equiv \quad \langle A[\langle B \rangle] \rangle$$

$$\text{NdAx4 } \langle \cdot [\phi] \rangle \quad \equiv \quad \phi$$

Note how little was needed to ensure booleanity!

(by Huntington's result from the 1930's)

And NdAx2–NdAx4 just mimic PrAx2—PrAx4
(redundancy: price to pay for two-sorted signature)

Axioms in one-sorted signature

Recall all the two-sorted axioms for predicates and expressions:

$$\text{PrAx1} \quad A[\neg\langle B \rangle] / B \equiv \perp$$

$$\text{PrAx2} \quad A[\phi \vee \psi] \equiv A[\phi] \cup A[\psi]$$

$$\text{PrAx3} \quad (A/B)[\phi] \equiv A/B[\phi]$$

$$\text{PrAx4} \quad \cdot[\langle \cdot \rangle] \equiv \cdot$$

$$\text{NdAx1} \quad \phi \equiv \neg(\neg\phi \vee \psi) \vee \neg(\neg\phi \vee \neg\psi)$$

$$\text{NdAx2} \quad \langle A \cup B \rangle \equiv \langle A \rangle \vee \langle B \rangle$$

$$\text{NdAx3} \quad \langle A/B \rangle \equiv \langle A[\langle B \rangle] \rangle$$

$$\text{NdAx4} \quad \langle \cdot[\phi] \rangle \equiv \phi$$

Axioms in one-sorted signature

Here is a one-sorted axiomatization for \sim over idempotent semi-ring axioms found by Hollenberg:

$$\begin{aligned}\sim A/A &\equiv \perp \\ \sim\sim A/A &\equiv A \\ \sim(A/B)/A &\equiv (\sim(A/B)/A)/\sim B \\ \sim(A \cup B) &\equiv \sim A/\sim B \\ \sim A \cup \sim B &\equiv \sim\sim(\sim A \cup \sim B)\end{aligned}$$

We need to add one more axiom for tests:

$$?p \equiv \sim\sim?p$$

Now, you may have the feeling that
there was nothing XPath-specific yet

Now, you may have the feeling that
there was nothing XPath-specific yet
But in fact there is a fragment for which
it is all there is:

Now, you may have the feeling that
there was nothing XPath-specific yet
But in fact there is a fragment for which
it is all there is:

Core XPath(\Downarrow), the child-axis-only fragment!

Theorem

The axioms presented so far are complete for all valid equivalences of Core XPath(\Downarrow).

Now, you may have the feeling that
there was nothing XPath-specific yet
But in fact there is a fragment for which
it is all there is:

Core XPath(\Downarrow), the child-axis-only fragment!

Theorem

The axioms presented so far are complete for all valid equivalences of Core XPath(\Downarrow).

In order to find more interesting equivalences,
we have to move to other fragments

Axioms for Linear Axes

The non-transitive case:

$$\text{LinAx1 } \mathbf{s}[\neg\phi] \equiv \cdot[\neg\langle\mathbf{s}[\phi]\rangle]/\mathbf{s} \text{ for } \mathbf{s} \in \{\Rightarrow, \Leftarrow, \Uparrow\}$$

This forces **functionality** of the corresponding axis

Axioms for Transitive Axes

One for node expressions, one for path expressions:

$$\begin{array}{l} \text{TransAx1} \quad \langle \mathbf{s}^+ [\phi] \rangle \quad \equiv \quad \langle \mathbf{s}^+ [\phi \wedge \neg \langle \mathbf{s}^+ [\phi] \rangle] \rangle \\ \text{TransAx2} \quad \mathbf{s}^+ \quad \equiv \quad \mathbf{s}^+ \cup \mathbf{s}^+ / \mathbf{s}^+ \end{array}$$

The first one is called the **Löb axiom** and forces well-foundedness
Don't get modal logicians started on it—
people wrote books about this formula

In particular, all the consequences of TransAx2 for *node expressions*
can be already derived from TransAx1
I can neither prove nor disprove that for *path expressions*
TransAx2 is (ir-)redundant

Finally, Axes which Are Both Transitive and Linear

$$\text{LinAx2} \cdot [\langle s^+ [\phi] \rangle] / s^+ \equiv s^+ [\phi] \cup s^+ [\phi] / s^+ \cup s^+ [\langle s^+ [\phi] \rangle]$$

for $s \in \{\Rightarrow, \Leftarrow, \Uparrow\}$

together with transitivity axioms

This forces the corresponding axis is a linear order

Single Axis Completeness Result

Theorem

- *Base axioms are complete for Core XPath(\Downarrow)*
- *Base axioms with LinAx1 are complete for other intransitive single axis fragments*
- *Base axioms with TransAx1 and TransAx2 are complete for Core XPath(\Downarrow^+)*
- *Base axioms with TransAx1, TransAx2 and LinAx2 are complete for other transitive single axis fragments*

A Few Words About Proofs

- First, rewrite node expressions to **simple node expressions**:

$$\text{siNode} ::= \langle \cdot \rangle \mid p \mid \langle a[\text{siNode}] \rangle \mid \neg \text{siNode} \mid \text{siNode} \vee \text{siNode}$$

A Few Words About Proofs

- First, rewrite node expressions to **simple node expressions**:

$\text{siNode} ::= \langle \cdot \rangle \mid p \mid \langle a[\text{siNode}] \rangle \mid \neg \text{siNode} \mid \text{siNode} \vee \text{siNode}$

They are isomorphic variants of **modal formulas**

A Few Words About Proofs

- First, rewrite node expressions to **simple node expressions**:

$$\text{siNode} ::= \langle \cdot \rangle \mid p \mid \langle a[\text{siNode}] \rangle \mid \neg \text{siNode} \mid \text{siNode} \vee \text{siNode}$$

They are isomorphic variants of **modal formulas**

- Using **normal form theorems** for modal logic, we provide a completeness proof for node expressions

A Few Words About Proofs cntd.

- Then we rewrite all path expressions as sums of sum-free expressions of the form

$$S = \cdot [\beta_1] / \mathbf{a} [\beta_2] / \dots / \mathbf{a} [\beta_\ell],$$

(all β_i are normal forms of

- the same **nesting degree** in case of transitive axes
- strictly decreasing degree for intransitive axes)

In case of linear axes, we can even guarantee that every formula is witnessed further down the chain

A Few Words About Proofs cntd.

- Then we rewrite all path expressions as sums of sum-free expressions of the form

$$S = \cdot [\beta_1] / \mathbf{a} [\beta_2] / \dots / \mathbf{a} [\beta_\ell],$$

(all β_i are normal forms of

- the same **nesting degree** in case of transitive axes
- strictly decreasing degree for intransitive axes)

In case of linear axes, we can even guarantee that every formula is witnessed further down the chain

- We prove that for every two such expressions either

A Few Words About Proofs cntd.

- Then we rewrite all path expressions as sums of sum-free expressions of the form

$$S = \cdot [\beta_1] / \mathbf{a} [\beta_2] / \dots / \mathbf{a} [\beta_\ell],$$

(all β_i are normal forms of

- the same **nesting degree** in case of transitive axes
- strictly decreasing degree for intransitive axes)

In case of linear axes, we can even guarantee that every formula is witnessed further down the chain

- We prove that for every two such expressions either
 - one is **a subsequence** of the other—**provably contained** or

A Few Words About Proofs cntd.

- Then we rewrite all path expressions as sums of sum-free expressions of the form

$$S = \cdot [\beta_1] / \mathbf{a} [\beta_2] / \dots / \mathbf{a} [\beta_\ell],$$

(all β_i are normal forms of

- the same **nesting degree** in case of transitive axes
- strictly decreasing degree for intransitive axes)

In case of linear axes, we can even guarantee that every formula is witnessed further down the chain

- We prove that for every two such expressions either
 - one is **a subsequence** of the other—**provably contained** or
 - there is **a countermodel** for containment

Aside: the issue of labels

There is a fact about XML trees we did not take into account
(unless we opt to render attribute-value pairs as additional labels)

Aside: the issue of labels

There is a fact about XML trees we did not take into account
(unless we opt to render attribute-value pairs as additional labels)

The labels are disjoint!

Aside: the issue of labels

There is a fact about XML trees we did not take into account
(unless we opt to render attribute-value pairs as additional labels)

The labels are disjoint!

However, this is easy to fix: add node axiom

$$p \wedge q \equiv \perp$$

for distinct p and q

This axiom itself is not substitution-invariant,
this is why we do not like it

Aside: the issue of labels

There is a fact about XML trees we did not take into account
(unless we opt to render attribute-value pairs as additional labels)

The labels are disjoint!

However, this is easy to fix: add node axiom

$$p \wedge q \equiv \perp$$

for distinct p and q

This axiom itself is not substitution-invariant,
this is why we do not like it

But as our proofs used only Birkhoff's rules
they are quite flexible and adding this axiom does not hurt

Axiom For Axes Dependencies

$$\begin{array}{l} \text{TreeAx1} \left\{ \begin{array}{l} \mathbf{s}^+ / \mathbf{s} \cup \mathbf{s} \equiv \mathbf{s}^+ \\ \mathbf{s} / \mathbf{s}^+ \cup \mathbf{s} \equiv \mathbf{s}^+ \end{array} \right. \\ \text{TreeAx2} \quad \mathbf{s}[\phi] / \mathbf{s}^\sim \equiv \cdot [\langle \mathbf{s}[\phi] \rangle] \text{ (for } \mathbf{s} \text{ distinct than } \uparrow) \\ \text{TreeAx3} \quad \uparrow[\phi] / \downarrow \equiv (\leftarrow^+ \cup \Rightarrow^+ \cup \cdot) [\langle \uparrow[\phi] \rangle] \\ \text{TreeAx4} \left\{ \begin{array}{l} \leftarrow^+ \equiv \leftarrow^+ [\langle \uparrow \rangle] \\ \Rightarrow^+ \equiv \Rightarrow^+ [\langle \uparrow \rangle] \end{array} \right. \end{array}$$

TreeAx1 says: \mathbf{s}^+ is a transitive closure of \mathbf{s}

TreeAx2 says non-child axes are functional
and describes their converse

TreeAx3 forces \uparrow is the converse of (non-functional) \downarrow
with TreeAx4, it also describes how horizontal and vertical axes
interplay

Theorem

*The axioms presented so far are complete for
Core XPath node expressions*

Theorem

*The axioms presented so far are complete for
Core XPath node expressions*

Proof.

By reduction to simple node expressions
and derivation of all axioms of **modal logic of finite trees**
by Blackburn, Meyer-Viol, de Rijke



(boolean axioms)


$$\begin{aligned}\langle \mathbf{s}[\neg(\cdot)] \rangle &\equiv \neg\langle \cdot \rangle \\ \langle \mathbf{s}[\phi \vee \psi] \rangle &\equiv \langle \mathbf{s}[\phi] \rangle \vee \langle \mathbf{s}[\psi] \rangle \\ \phi &\leq \neg\langle \mathbf{s}[\neg\langle \mathbf{s}^\sim[\phi] \rangle] \rangle \\ \langle \mathbf{s}[\neg\phi] \rangle \wedge \langle \mathbf{s}[\phi] \rangle &\equiv \neg\langle \cdot \rangle \text{ (for } \mathbf{s} \text{ distinct than } \uparrow) \\ \langle \mathbf{s}[\phi] \rangle \vee \langle \mathbf{s}[\langle \mathbf{s}^+[\phi] \rangle] \rangle &\equiv \langle \mathbf{s}^+[\phi] \rangle \\ \neg\langle \mathbf{s}[\phi] \rangle \wedge \langle \mathbf{s}^+[\phi] \rangle &\leq \langle \mathbf{s}^+[\neg\phi \wedge \langle \mathbf{s}[\phi] \rangle] \rangle \\ \langle \mathbf{s}[\langle \cdot \rangle] \rangle &\leq \langle \mathbf{s}^+[\neg\langle \mathbf{s}[\langle \cdot \rangle] \rangle] \rangle \\ \text{TransAx1 for } \downarrow^+ \text{ and } \Rightarrow^+ & \\ \langle \downarrow[\neg\langle \Leftarrow \rangle \wedge \neg\langle \Rightarrow^*[\phi] \rangle] \rangle &\leq \neg\langle \downarrow[\phi] \rangle \\ \langle \downarrow[\phi] \rangle &\leq \langle \downarrow[\neg\langle \Leftarrow \rangle] \rangle \wedge \langle \downarrow[\neg\langle \Rightarrow \rangle] \rangle \\ \neg\langle \uparrow \rangle &\leq \neg\langle \Leftarrow \rangle \wedge \neg\langle \Rightarrow \rangle\end{aligned}$$

The Nasty Trick

The Nasty Trick


We can use this to provide
an axiomatization for path expressions . . .

The Nasty Trick

We can use this to provide
an axiomatization for path expressions ...
... of a sort—a non-orthodox one! 

The Nasty Trick

We can use this to provide
an axiomatization for path expressions ...


... of a sort—a non-orthodox one! 

Add the separability rule:

(Sep) IF $\langle A[p] \rangle \equiv \langle B[p] \rangle$ for p not occurring in A, B
THEN $A \equiv B$.

The Nasty Trick

We can use this to provide
an axiomatization for path expressions ...

... of a sort—a non-orthodox one! 

Add the separability rule:

(Sep) IF $\langle A[p] \rangle \equiv \langle B[p] \rangle$ for p not occurring in A, B

THEN $A \equiv B$.

Except for spoiling the whole equational story,
it does not sit too well with the labelling axiom ...

The Nasty Trick Does Its Job

... but it's perfect for obtaining **complexity** results
for **query equivalence** problem
by using **reductions to corresponding modal logics**

Theorem

- Query equivalence of Core XPath(\Rightarrow^+ , \Leftarrow^+), Core XPath(\Uparrow^+), Core XPath(\mathbf{s}) (for $\mathbf{s} \in \{\Uparrow, \Leftarrow, \Rightarrow\}$) is coNP-complete.

- Query equivalence of Core XPath(\Leftarrow^+ , \Leftarrow , \Rightarrow^+ , \Rightarrow , \Uparrow^+ , \Uparrow) is PSPACE-complete.

Thus, the PSPACE upper bound applies to all its sublanguages.

- Query equivalence of Core XPath(\Downarrow) and Core XPath(\Downarrow^+) is PSPACE-complete.

Thus, all extensions of this fragment are PSPACE-hard.

- Query equivalence of Core XPath(\Downarrow , \Downarrow^+) is EXPTIME-complete.

Thus, all extensions of this fragment are EXPTIME-hard.

... by reductions to complexity results for modal logics like **K**, **GL**, **Alt.1** and **fragments of tense/temporal logic on linear and branching orders**.

The most interesting one is for the second clause—somewhat tricky embedding into a logic of Sistla and Clarke

What we have seen so far . . .

We have seen:

- **equational** axiomatizations for **path equivalences** of all eight **single axis fragments** of Core XPath

What we have seen so far . . .

We have seen:

- **equational** axiomatizations for **path equivalences** of all eight **single axis fragments** of Core XPath
- **equational** axiomatizations for **node equivalences** of **full** Core XPath 1.0

What we have seen so far . . .

We have seen:

- **equational** axiomatizations for **path equivalences** of all eight **single axis fragments** of Core XPath
- **equational** axiomatizations for **node equivalences** of full Core XPath 1.0
- **non-orthodox** axiomatization for **path equivalences** of full Core XPath 1.0

What we have seen so far . . .

We have seen:

- **equational** axiomatizations for **path equivalences** of all eight **single axis fragments** of Core XPath
- **equational** axiomatizations for **node equivalences** of full Core XPath 1.0
- **non-orthodox** axiomatization for **path equivalences** of full Core XPath 1.0
- **computational complexity results** for path equivalences in most meaningful sublanguages of Core XPath 1.0

*What we have **not** seen so far . . .*

- **Definability** and **expressivity** results
- Results for **stronger** fragments of XPath

Expressive power

Possible yardsticks for expressive power on trees:

- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... — e.g., in between FO and MSO lies FO(TC)

Expressive power

Possible yardsticks for expressive power on trees:

- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... — e.g., in between FO and MSO lies FO(TC)

What kind of queries do we want to characterize

Possible yardsticks for expressive power on trees:

- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... — e.g., in between FO and MSO lies FO(TC)

What kind of queries do we want to characterize

- **Binary relations** definable by **path expressions**?
- **Node sets** definable by **node expressions**?
- **Properties of trees** definable by **node expressions** evaluated at the root?

Expressive power

Possible yardsticks for expressive power on trees:

- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... — e.g., in between FO and MSO lies FO(TC)

What kind of queries do we want to characterize

- **Binary relations** definable by **path expressions**?
- **Node sets** definable by **node expressions**?
- **Properties of trees** definable by **node expressions evaluated at the root**?

Possible types of characterizations:

Possible yardsticks for expressive power on trees:

- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... — e.g., in between FO and MSO lies FO(TC)

What kind of queries do we want to characterize

- **Binary relations** definable by **path expressions**?
- **Node sets** definable by **node expressions**?
- **Properties of trees** definable by **node expressions evaluated at the root**?

Possible types of characterizations:

- **Syntactic** (e.g. “ L is equivalent to the two variable ...”) versus **semantic** (e.g., “bisimulation invariant fragment ...”)

Possible yardsticks for expressive power on trees:

- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... — e.g., in between FO and MSO lies FO(TC)

What kind of queries do we want to characterize

- **Binary relations** definable by **path expressions**?
- **Node sets** definable by **node expressions**?
- **Properties of trees** definable by **node expressions evaluated at the root**?

Possible types of characterizations:

- **Syntactic** (e.g. “ L is equivalent to the two variable ...”) versus **semantic** (e.g., “bisimulation invariant fragment ...”)

Decidable characterizations?

Descendant-only fragment

$\text{CoreXPath}(\downarrow^+)$ **node** expressions have the same expressive power as MSO formulas $\varphi(x)$ for which

- (i) truth of $\varphi(x)$ at a node depends only on the subtree
- (ii) $\varphi(x)$ does not distinguish children from descendants, i.e., the following operation preserves truth/falsity at the root:



Descendant-only fragment

$\text{CoreXPath}(\Downarrow^+)$ **node** expressions have the same expressive power as MSO formulas $\varphi(x)$ for which

- (i) truth of $\varphi(x)$ at a node depends only on the subtree
- (ii) $\varphi(x)$ does not distinguish children from descendants, i.e., the following operation preserves truth/falsity at the root:



Easy **proof** from **De Jongh-Sambin fixed point theorem for GL**
and **Janin-Walukiewicz theorem for μ -calculus**

Descendant-only fragment

$\text{CoreXPath}(\downarrow^+)$ **node** expressions have the same expressive power as MSO formulas $\varphi(x)$ for which

- (i) truth of $\varphi(x)$ at a node depends only on the subtree
- (ii) $\varphi(x)$ does not distinguish children from descendants, i.e., the following operation preserves truth/falsity at the root:



Easy **proof** from **De Jongh-Sambin fixed point theorem for GL**
and **Janin-Walukiewicz theorem for μ -calculus**

Moreover, the proof is **effective**: it yields a decision procedure.

Descendant-only fragment

$\text{CoreXPath}(\downarrow^+)$ **node** expressions have the same expressive power as MSO formulas $\varphi(x)$ for which

- (i) truth of $\varphi(x)$ at a node depends only on the subtree
- (ii) $\varphi(x)$ does not distinguish children from descendants, i.e., the following operation preserves truth/falsity at the root:



Easy **proof** from **De Jongh-Sambin fixed point theorem for GL**
and **Janin-Walukiewicz theorem for μ -calculus**

Moreover, the proof is **effective**: it yields a decision procedure.

A variant for **path** expressions obtained by replacing

μ -**formulas** by μ -**programs**
bisimulation **invariance** by bisimulation **safety**

Expressive power (ct'd)

What about the full **Core XPath** language?

What about the full **Core XPath** language?

No decidable characterization in terms of MSO is known.
(but see a LICS 2010 result of Place/Segoufin for a decidable
characterization of *CoreXPath*(\Downarrow^+ , \Uparrow^+ , \Leftarrow^+ , \Rightarrow^+) in terms of
forest algebras)

What about the full **Core XPath** language?

No decidable characterization in terms of MSO is known.
(but see a LICS 2010 result of Place/Segoufin for a decidable characterization of *CoreXPath*(\downarrow^+ , \uparrow^+ , \Leftarrow^+ , \Rightarrow^+) in terms of **forest algebras**)

Syntactic characterization of full CXP (Marx-De Rijke 05)

Core XPath node expressions have the same expressive power as formulas $\phi(x)$ in the **two-variable fragment** of $FO[R_{\downarrow}, R_{\downarrow^+}, R_{\Rightarrow}, R_{\Rightarrow^+}]$.

There is a similar characterization for **path expressions**.

In summary...

- Core XPath is **reasonably expressive** yet **computationally attractive**.

In summary...

- Core XPath is **reasonably expressive** yet **computationally attractive**.
- In the remainder of this talk, we consider two extensions:

In summary...

- Core XPath is **reasonably expressive** yet **computationally attractive**.
- In the remainder of this talk, we consider two extensions:
 - 1 **Regular XPath**: the extension of Core XPath with full transitive closure.

In summary...

- Core XPath is **reasonably expressive** yet **computationally attractive**.
- In the remainder of this talk, we consider two extensions:
 - 1 **Regular XPath**: the extension of Core XPath with full **transitive closure**.
 - 2 **Core XPath 2.0**: the navigational core of XPath 2.0, featuring **path intersection and complementation** and more.

Motivation for Regular XPath

Extending XPath with transitive closure proposed for various reasons.

Motivation for Regular XPath

Extending XPath with transitive closure proposed for various reasons.

- Practical reasons: some applications require the use of transitive closure.

(an example: Nentvich et al. 2002, `xlinkit`: a consistency checking and smart link generation service. ACM Trans. on Int. Tech. 2002)

Motivation for Regular XPath

Extending XPath with transitive closure proposed for various reasons.

- Practical reasons: some applications require the use of transitive closure.

(an example: Nentvich et al. 2002, `xlinkit`: a consistency checking and smart link generation service. ACM Trans. on Int. Tech. 2002)

- Core XPath extended with transitive closure
 - has full first-order expressive power
 - is rich enough to express DTDs and
 - admits view-based query rewriting with recursive views

Motivation for Regular XPath

Extending XPath with transitive closure proposed for various reasons.

- Practical reasons: some applications require the use of transitive closure.
(an example: Nentvich et al. 2002, `xlinkit`: a consistency checking and smart link generation service. ACM Trans. on Int. Tech. 2002)
- Core XPath extended with transitive closure
 - has full first-order expressive power
 - is rich enough to express DTDs and
 - admits view-based query rewriting with recursive views
- very natural from the perspective of PDL

Motivation for Regular XPath

Extending XPath with transitive closure proposed for various reasons.

- Practical reasons: some applications require the use of transitive closure.

(an example: Nentvich et al. 2002, `xlinkit`: a consistency checking and smart link generation service. ACM Trans. on Int. Tech. 2002)

- Core XPath extended with transitive closure
 - has full first-order expressive power
 - is rich enough to express DTDs and
 - admits view-based query rewriting with recursive views
- very natural from the perspective of PDL

- 1 Part of community standard EXSLT
- 2 Implemented (naively) in Saxon and Xalan
- 3 ... but left out by W3C from XPath 2.0

Syntax of Regular XPath

- Regular XPath has two types of expressions:

Syntax of Regular XPath

- Regular XPath has two types of expressions:

- **path expressions**

$\alpha ::= \uparrow \mid \downarrow \mid \leftarrow \mid \Rightarrow \mid \cdot \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha^* \mid \alpha[\phi]$

Syntax of Regular XPath

- Regular XPath has two types of expressions:

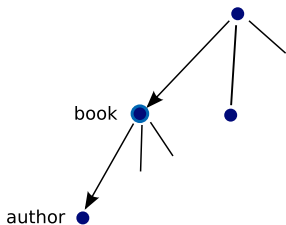
- **path expressions**

$\alpha ::= \uparrow \mid \downarrow \mid \leftarrow \mid \rightarrow \mid \cdot \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha^* \mid \alpha[\phi]$

- **node expressions**

$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle \alpha \rangle$

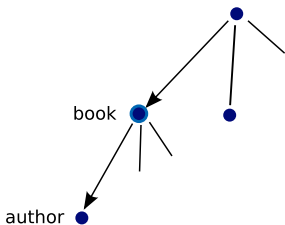
An example



“Go to the next book that has at least two authors.”

In Regular XPath:

An example



“Go to the next book that has at least two authors.”

In Regular XPath:

$$(\Rightarrow[\neg twoauthorbook])^* / \Rightarrow[twoauthorbook]$$

where *twoauthorbook* stands for
 $book \wedge \langle \downarrow[author] / \Rightarrow^+[author] \rangle$.

Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

- Let $(\alpha \textit{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$.

Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

- Let $(\alpha \textit{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$.
- Let **root** be short for $\neg\langle\uparrow\rangle$.
Let **leaf** be short for $\neg\langle\downarrow\rangle$.
Let **first** be short for $\neg\langle\leftarrow\rangle$.
Let **last** be short for $\neg\langle\rightarrow\rangle$.

Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

- Let $(\alpha \textit{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$.
- Let **root** be short for $\neg\langle\uparrow\rangle$.
Let **leaf** be short for $\neg\langle\downarrow\rangle$.
Let **first** be short for $\neg\langle\leftarrow\rangle$.
Let **last** be short for $\neg\langle\rightarrow\rangle$.
- Let **suc** be shorthand for $\downarrow[\textit{first}] \cup .[\textit{leaf}]/(\uparrow \textit{ while } \textit{last})/\Rightarrow$
(the successor in depth first left-to-right ordering).

Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

- Let $(\alpha \textit{ while } \phi)$ be shorthand for $(.[\phi]/\alpha)^*$.
- Let **root** be short for $\neg\langle\uparrow\rangle$.
Let **leaf** be short for $\neg\langle\downarrow\rangle$.
Let **first** be short for $\neg\langle\leftarrow\rangle$.
Let **last** be short for $\neg\langle\rightarrow\rangle$.
- Let **suc** be shorthand for $\downarrow[\textit{first}] \cup .[\textit{leaf}]/(\uparrow \textit{ while } \textit{last})/\Rightarrow$
(the successor in depth first left-to-right ordering).
- Then $\langle (suc/suc)^*[\textit{leaf}]/(\uparrow \textit{ while } \textit{last})[\textit{root}] \rangle$ is true at the root iff the number of nodes is even.

One more example

- Consider game trees:
 - **leafs** are labeled by **Anne-wins** or **Bob-wins**
 - **inner nodes** are labeled by **Anne's-move** or **Bob's-move**

One more example

- Consider game trees:
 - **leafs** are labeled by **Anne-wins** or **Bob-wins**
 - **inner nodes** are labeled by **Anne's-move** or **Bob's-move**
- Puzzle:
Show that “**Anne has a winning strategy**” is expressible.

Expressive power of Regular XPath

Expressive power of Regular XPath

- What is the expressive power of **Regular XPath**?
- We know that

$$FO \subsetneq \text{Regular XPath} \subseteq FO(TC)$$

(The first inclusion follows from results by Marx 2004).

Expressive power of Regular XPath

- What is the expressive power of **Regular XPath**?
- We know that

$$FO \subsetneq \text{Regular XPath} \subseteq FO(TC)$$

(The first inclusion follows from results by Marx 2004).

- A natural conjecture:

$$\text{Regular XPath} \equiv FO(TC)$$

(after all, Regular XPath **has** a transitive closure operator!)

Expressive power of Regular XPath

- What is the expressive power of **Regular XPath**?
- We know that

$$FO \subsetneq \text{Regular XPath} \subseteq FO(TC)$$

(The first inclusion follows from results by Marx 2004).

- A natural conjecture:

$$\text{Regular XPath} \equiv FO(TC)$$

(after all, Regular XPath **has** a transitive closure operator!)

- Ten Cate and Segoufin managed to prove this only after extending Regular XPath with a “**within**” operator **W**:

$$T, n \models W\phi \quad \text{iff} \quad T_n, n \models \phi$$

(cf. temporal logics with forgettable past)

Expressive power of Regular XPath (ct'd)

Expressive power of Regular XPath (ct'd)

- $FO(TC)$ is the extension of first-order logic with a transitive closure operator for binary relations.

Expressive power of Regular XPath (ct'd)

- $FO(TC)$ is the extension of first-order logic with a transitive closure operator for binary relations.

Theorem: (Ten Cate and Segoufin, PODS 2008, JACM 2010)

Regular XPath(W) path expressions define the same binary relations as $FO(TC)$ formulas with two free variables. Similarly for node expressions.

Expressive power of Regular XPath (ct'd)

- $FO(TC)$ is the extension of first-order logic with a transitive closure operator for binary relations.

Theorem: (Ten Cate and Segoufin, PODS 2008, JACM 2010)

Regular XPath(W) path expressions define the same binary relations as $FO(TC)$ formulas with two free variables. Similarly for node expressions.

- **Corollary:** Regular XPath(W) is closed under path intersection and complementation.

Axiomatizations and complexity

No axiomatizations are known yet for Regular XPath and Regular XPath(W).

Axiomatizations and complexity

No axiomatizations are known yet for Regular XPath and Regular XPath(W).

As for complexity,

Axiomatizations and complexity

No axiomatizations are known yet for Regular XPath and Regular XPath(W).

As for complexity,

- Query evaluation can still be performed in **PTime** even for **Regular XPath(W)**.
- Query containment is still **ExpTime-complete** for **Regular XPath** but it is **2ExpTime-complete** for **Regular XPath(W)**

Core XPath 2.0

XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection** and **complementation** of path expressions.

α intersect β **and** α except β

XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection** and **complementation** of path expressions.

α intersect β **and** α except β

Example: $\Downarrow^*[p]$ except $\Downarrow^*[q] / \Downarrow^*[p]$

XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection** and **complementation** of path expressions.

α intersect β **and** α except β

Example: $\Downarrow^*[p]$ except $\Downarrow^*[q] / \Downarrow^*[p]$

- **Variables** and **for loops**

for $\$x$ in α return β **and**

$\alpha[. \text{ is } \$x]$

XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection** and **complementation** of path expressions.

α intersect β and α except β

Example: $\Downarrow^*[p]$ except $\Downarrow^*[q] / \Downarrow^*[p]$

- **Variables** and **for loops**

for $\$x$ in α return β and
 $\alpha[. \text{ is } \$x]$

Example:

for $\$x$ in $.$ return $\Downarrow^*[p] \wedge \neg \left\langle \Uparrow^*[q] / \Uparrow^* [. \text{ is } \$x] \right\rangle$

XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection** and **complementation** of path expressions.

α intersect β and α except β

Example: $\Downarrow^*[p]$ except $\Downarrow^*[q] / \Downarrow^*[p]$

- **Variables** and **for loops**

for $\$x$ in α return β and
 $\alpha[. \text{ is } \$x]$

Example:

for $\$x$ in $.$ return $\Downarrow^*[p \wedge \neg \langle \Uparrow^*[q] / \Uparrow^* [. \text{ is } \$x] \rangle]$

- Core XPath 2.0 is the extension of Core XPath with these features.

- The **path intersection and complementation** turn Core XPath 2.0 into a version of Tarski's relation algebra (interpreted on finite ordered trees).

- The **path intersection and complementation** turn Core XPath 2.0 into a version of Tarski's relation algebra (interpreted on finite ordered trees).
- The **variables and for-loops** make it possible to give a linear translation from first-order logic to Core XPath 2.0:

$$TR(\phi(x, y)) = \text{for } \$x \text{ in } ., \$y \text{ in } \top \text{ return } \$y[TR'(\phi)]$$

$$TR'(x = y) = \langle \top[. \text{ is } \$x \wedge . \text{ is } \$y] \rangle$$

$$TR'(R_{\downarrow}xy) = \langle \top[. \text{ is } \$x \wedge \langle \downarrow[. \text{ is } \$y] \rangle] \rangle$$

$$TR'(R_{\downarrow^*}xy) = \langle \top[. \text{ is } \$x \wedge \langle \downarrow^*[. \text{ is } \$y] \rangle] \rangle$$

$$TR'(R_{\Rightarrow}xy) = \langle \top[. \text{ is } \$x \wedge \langle \Rightarrow[. \text{ is } \$y] \rangle] \rangle$$

$$TR'(R_{\Rightarrow^*}xy) = \langle \top[. \text{ is } \$x \wedge \langle \Rightarrow^*[. \text{ is } \$y] \rangle] \rangle$$

$$TR'(\phi \wedge \psi) = TR'(\phi) \wedge TR'(\psi)$$

$$TR'(\neg\phi) = \neg TR'(\phi)$$

$$TR'(\exists x.\phi) = \text{for } \$x \text{ in } \top \text{ return } TR'(\phi)$$

where \top is shorthand for \uparrow^*/\downarrow^* (the universal relation)

Expressivity, Complexity and Axiomatization

- Core XPath 2.0 has the **same expressive power as first-order logic**, both with and without variables (in the case with variables there is a linear translation).

Expressivity, Complexity and Axiomatization

- Core XPath 2.0 has the **same expressive power as first-order logic**, both with and without variables (in the case with variables there is a linear translation).
- The complexity of the **query equivalence** problem is **non-elementary**, both with and without variables.

Expressivity, Complexity and Axiomatization

- Core XPath 2.0 has the **same expressive power as first-order logic**, both with and without variables (in the case with variables there is a linear translation).
- The complexity of the **query equivalence** problem is **non-elementary**, both with and without variables.
(even adding only **path intersection** to Core XPath makes it 2ExpTime-complete.)

Expressivity, Complexity and Axiomatization

- Core XPath 2.0 has the **same expressive power as first-order logic**, both with and without variables (in the case with variables there is a linear translation).
- The complexity of the **query equivalence** problem is **non-elementary**, both with and without variables.

(even adding only **path intersection** to Core XPath makes it 2ExpTime-complete.)
- We have two **complete axiomatizations** of path equivalence in Core XPath 2.0: one with and one without variables.

The case without variables

- Recall that, without variables, Core XPath 2.0 is essentially a version of **Relation Algebra** interpreted on **finite sibling ordered trees**.

The case without variables

- Recall that, without variables, Core XPath 2.0 is essentially a version of **Relation Algebra** interpreted on **finite sibling ordered trees**.
- One apparent problem: Relation Algebra has no node tests.

However, these can easily be translated away:

$$\begin{aligned} \textit{Pred1.} \quad \alpha[\phi \wedge \psi] &\equiv \alpha[\phi][\psi] \\ \textit{Pred2.} \quad \alpha[\phi \vee \psi] &\equiv \alpha[\phi] \cup \alpha[\psi] \\ \textit{Pred3.} \quad \alpha[\neg\phi] &\equiv \alpha - \alpha[\phi] \\ \textit{Pred4.} \quad \alpha[\langle\beta\rangle] &\equiv \alpha / ((\beta/\top) \cap .) \end{aligned}$$

The case without variables

- Recall that, without variables, Core XPath 2.0 is essentially a version of **Relation Algebra** interpreted on **finite sibling ordered trees**.
- One apparent problem: Relation Algebra has no node tests.

However, these can easily be translated away:

$$\begin{aligned} \text{Pred1. } \alpha[\phi \wedge \psi] &\equiv \alpha[\phi][\psi] \\ \text{Pred2. } \alpha[\phi \vee \psi] &\equiv \alpha[\phi] \cup \alpha[\psi] \\ \text{Pred3. } \alpha[\neg\phi] &\equiv \alpha - \alpha[\phi] \\ \text{Pred4. } \alpha[\langle\beta\rangle] &\equiv \alpha / ((\beta/\top) \cap .) \end{aligned}$$

- Besides these axioms, our axiomatization for variable free Core XPath 2.0 contains **two groups of axioms**:
 - General axioms of Boolean Algebra and Relation Algebra
 - Axioms describing (first-order) properties of trees.

Axioms of Boolean algebra

- BA1. $\alpha \cup (\beta \cap \gamma) \equiv (\alpha \cup \beta) \cap (\alpha \cup \gamma)$
- BA2. $\alpha \cap (\beta \cup \gamma) \equiv (\alpha \cap \beta) \cup (\alpha \cap \gamma)$
- BA3. $\alpha \cup \beta \equiv \beta \cup \alpha$
- BA4. $\alpha \cap \beta \equiv \beta \cap \alpha$
- BA5. $\alpha \cup (\beta \cap \gamma) \equiv (\alpha \cup \beta) \cap (\alpha \cup \gamma)$
- BA6. $\alpha \cap (\beta \cup \gamma) \equiv (\alpha \cap \beta) \cup (\alpha \cap \gamma)$
- BA7. $\alpha \cup (\alpha \cap \beta) \equiv \alpha$
- BA8. $\alpha \cap (\alpha \cup \beta) \equiv \alpha$
- BA9. $\alpha \cup (\top - \alpha) \equiv \top$
- BA10. $\alpha \cap (\top - \alpha) \equiv \perp$
- BA11. $\alpha \cap (\top - \beta) \equiv \alpha - \beta$

The axioms of Relation algebra

- RA1. $\alpha/(\beta/\gamma) \equiv (\alpha/\beta)/\gamma$
 - RA2. $\alpha/. \equiv \alpha$
 - RA3. $(\alpha \cup \beta)/\gamma \equiv \alpha/\gamma \cup \beta/\gamma$
 - RA4. $(\alpha \cup \beta)^\smile \equiv \alpha^\smile \cup \beta^\smile$
 - RA5. $(\alpha/\beta)^\smile \equiv \beta^\smile/\alpha^\smile$
 - RA6. $(\alpha^\smile)^\smile \equiv \alpha$
 - RA7. $(\alpha/(\top - (\alpha^\smile/\beta))) \subseteq \top$ except β
- To completely axiomatize relation algebra, normally, one needs to add also **Venema's Rule**:
- If X is a relation variable not occurring in α and $X - (((\top - .)/X/\top) \cup (\top/X/(\top - .))) \subseteq \alpha$ then $\alpha \equiv \top$.*

Fortunately, this rule turns out to be derivable in our case.

The axioms for finite sibling ordered trees

Tr1.	$\Downarrow^+ / \Downarrow^+$	\subseteq	\Downarrow^+
Tr2.	$\Downarrow^+ \cap \Uparrow^+$	\equiv	\perp
Tr3a.	\Downarrow^+	\equiv	$\Downarrow \cup (\Downarrow / \Downarrow^+)$
Tr3b.	\Downarrow	\equiv	$\Downarrow^+ - (\Downarrow^+ / \Downarrow^+)$
Tr4.	$\cdot[\langle \Uparrow \rangle]$	\equiv	$\cdot[\Uparrow^+[\neg \langle \Uparrow \rangle]]$
Tr5.	$\Downarrow^+ / \Uparrow^+$	\equiv	$\Downarrow^+ \cup \cdot[\langle \Downarrow \rangle] \cup \cdot[\langle \Downarrow \rangle] / \Uparrow^+$
Tr6.	$\Rightarrow^+ / \Rightarrow^+$	\subseteq	\Rightarrow^+
Tr7.	$\Rightarrow^+ \cap \Leftarrow^+$	\equiv	\perp
Tr8a.	\Rightarrow^+	\equiv	$\Rightarrow \cup (\Rightarrow / \Rightarrow^+)$
Tr8b.	\Rightarrow	\equiv	$\Rightarrow^+ - (\Rightarrow^+ / \Rightarrow^+)$
Tr9.	$\cdot[\langle \Leftarrow \rangle]$	\equiv	$\cdot[\langle \Leftarrow^+[\neg \langle \Leftarrow \rangle]]]$
Tr10.	$\Rightarrow^+ \cup \Leftarrow^+$	\equiv	$(\Uparrow / \Downarrow) - \cdot$
Tr11.	$\cdot \cup \Uparrow^+ \cup \Downarrow^+ \cup$ $(\Uparrow^* / \Rightarrow^+ / \Downarrow^*) \cup (\Uparrow^* / \Leftarrow^+ / \Downarrow^*)$	\equiv	\top
Ind.	$\top[\langle \alpha \rangle]$	\equiv	$\top[\langle \alpha - (\alpha / \Leftarrow \Leftarrow) \rangle]$

Three languages:

Three languages:

- **Core XPath**: the navigational core of XPath 1.0
Expressivity: FO^2
Query evaluation: PTime
Query equivalence: ExpTime-complete

Three languages:

- **Core XPath**: the navigational core of XPath 1.0
Expressivity: FO^2
Query evaluation: PTime
Query equivalence: ExpTime-complete
- **Regular XPath(W)**: the extension with $*$ and W .
Expressivity: same as $FO(TC)$
Query evaluation: PTime
Query equivalence: 2ExpTime-complete

Three languages:

- **Core XPath**: the navigational core of XPath 1.0
Expressivity: FO^2
Query evaluation: PTime
Query equivalence: ExpTime-complete
- **Regular XPath(W)**: the extension with $*$ and W .
Expressivity: same as $FO(TC)$
Query evaluation: PTime
Query equivalence: $2ExpTime$ -complete
- **Core XPath 2.0**: the navigational core of XPath 2.0
Expressivity: same as FO.
Query evaluation: PSpace-complete
Query equivalence: non-elementary hard

Expressive power:

Marx and De Rijke. *Semantic characterizations of navigational XPath*. SIGMOD Record 34(2), 2005

Ten Cate, Fontaine and Litak. *Some modal aspects of XPath*. M4M'07. Journal version to appear in a special "20th birthday" issue of JANCL

Ten Cate and Segoufin. *XPath, transitive closure logic, and nested tree walking automata*. Journal of the ACM, 2010. Extended abstract appeared in PODS 2008.

Place/Segoufin in LICS 2010, Fontaine/Place in MFCS 2010 ...

Axiomatization:

Ten Cate, Fontaine and Litak. *Some modal aspects of XPath*. M4M'07. Journal version to appear in a special "20th birthday" issue of JANCL

Ten Cate, Litak and Marx. Complete axiomatizations of XPath fragments. JAL 2010. Extended abstract presented at LiD 2008.

Ten Cate and Marx. *Axiomatizing the logical core of XPath 2.0*. ICDT'07.

Complexity:

Gottlob, Koch and Pichler. *Efficient algorithms for processing XPath queries*. TODS 30(2), 2005

Ten Cate and Lutz. *Query containment in very expressive XPath dialects*. PODS'07.