Queries, Modalities, Relations, Trees, XPath
Lecture VI
Harvest: Core XPath 1.0
as a Modal Logic for Trees
Axiomatizations and Complexity

Tadeusz Litak

Department of Computer Science
University of Leicester

July 2010: draft

Before we begin . . .

I am sometimes asked by students (not researchers, as they know the answer)

why theory is needed?
is it possible to just do applications
without any theory?

The answer

was provided by

Charles-Louis de Secondat,
baron de La Bréde et de Montesquieu,
one of the greatest European philosophers of law
(XVIIIth century)

An Idea of Despotic Power

When the savages of Louisiana are desirous of fruit,

they cut the tree to the root, and gather the fruit.

This is an emblem of despotic government.

Trying to learn "applications" without necessary background

means **cutting the tree of learning**

In the long run, you are not going to have any fruits

Only after you learn theory properly, you realize

Only after you learn theory properly, you realize

- how many results you can produce

Only after you learn theory properly, you realize

- how many results you can produce
- how closely things are connected

Only after you learn theory properly, you realize

- how many results you can produce
- how closely things are connected
- how many tools and techniques from the past you can reuse

FROM NOW ON, THE LECTURE IS MEANT TO ILLUSTRATE THIS

To use José's terminology, the contents belong

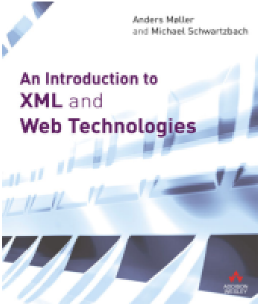
to Web Services Architecture

and more specifically, of course,

to XML Technologies

XML and Web Technologies

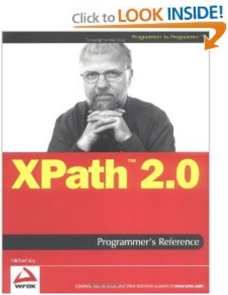
A good overall reference:



Webpage for the book: *http://www.brics.dk/ixwt/*

XPath

Most detailed reference on XPath
(except for W3C specification itself):



XML and Semi-structured Data

XML and Semi-structured Data

XML

eXtensible Markup Language

XML and Semi-structured Data

XML

eXtensible Markup Language

- began as a subset of SGML:

  Standard Generalized Markup Language

  (HTML: simplified and corrupted subset of SGML)

XML and Semi-structured Data

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 and Semi-structured Data

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 and Semi-structured Data

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 and Semi-structured Data

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:

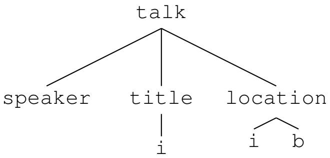
```
<?xml version='1.0' encoding='UTF-8'?>
<talk date='23-Jul-2010'>
  <speaker uni='Leicester'>T. Litak</speaker>
  <title>
    <i>Core XPath 1.0</i>as a Modal Logic
  </title>
  <location>
    <i>JXNU</i><b>Nanchang</b>
  </location>
</talk>
```

(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

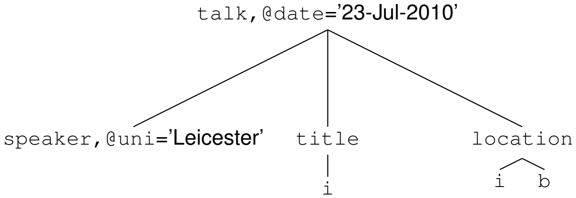
Either this . . .



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

What we'll see through our dim glasses

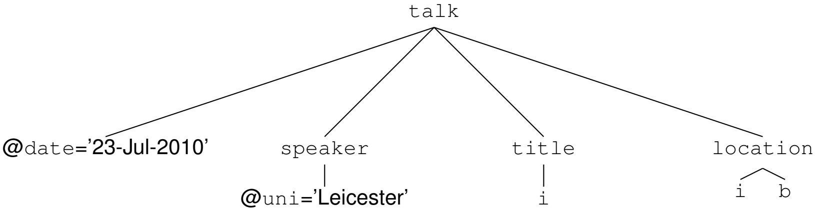
or that ...



(attribute-value pairs are additional labels)

What we'll see through our dim glasses

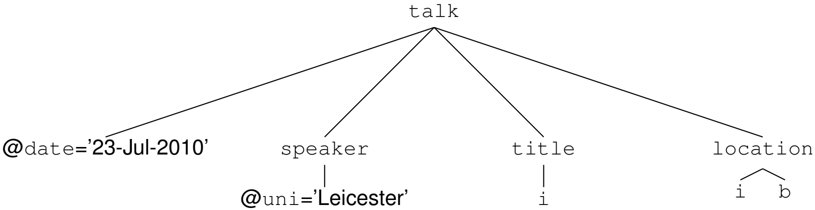
or perhaps . . .



(back to the unique labelling idea, attribute-value pairs are a special kind of children)

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)

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**. . . .
- . . .
- Specification of XPath 1.0 (W3C, Nov ’99): $\pm$ 30 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**. ...
- ...
- Specification of XPath 1.0 (W3C, Nov '99): $\pm$ 30 pages.
- Specification of XPath 2.0 (W3C, Nov '05): $\pm$ 90 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**. ...
- ...
- Specification of XPath 1.0 (W3C, Nov '99): $\pm$ 30 pages.
- Specification of XPath 2.0 (W3C, Nov '05): $\pm$ 90 pages.
- Specification of XPath 3.0: $\pm$ 270 pages? (Balder's extrapolation)

Core XPath 1.0

We focus on the basic navigational functionality of XPath:

Core XPath 1.0

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

Core XPath 1.0

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

Core XPath 1.0

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

An additional advantage of such a simple language:

data model discrepancies between XPath 1.0 and 2.0
no longer relevant

Core XPath

Core XPath has two types of expressions:

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

Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

$\mathsf{s} ::= \downarrow, \uparrow, \leftarrow, \rightarrow$
$\mathsf{a} ::= \mathsf{s} \mid \mathsf{s}^{+}$
$\mathsf{pexpr} ::= \mathsf{a} \mid \cdot \mid \mathsf{pexpr}/\mathsf{pexpr} \mid \mathsf{pexpr} \cup \mathsf{pexpr} \mid \mathsf{pexpr}[\mathsf{nexpr}]$

Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

$$s ::= \downarrow, \uparrow, \leftarrow, \rightarrow$$
$$a ::= s \mid s^{+}$$
$$\text{pexpr} ::= a \mid \cdot \mid \text{pexpr}/\text{pexpr} \mid \text{pexpr} \cup \text{pexpr} \mid \text{pexpr}[\text{nexpr}]$$
$$\text{nexpr} ::= p \mid \langle\text{pexpr}\rangle \mid \neg\text{nexpr} \mid \text{nexpr} \vee \text{nexpr} \qquad (p \in \Sigma)$$

Core XPath

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]`

$\mathrm{a} ::= \mathrm{s} \mid \mathrm{s}^{+}$

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

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

Core XPath

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::⋆
```

pexpr ::= a | $\cdot$ | pexpr/pexpr | pexpr $\cup$ pexpr | pexpr[nexpr]
nexpr ::= p | $\langle$pexpr$\rangle$ | $\neg$nexpr | nexpr $\vee$ nexpr $\quad (p \in \Sigma)$

Core XPath

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 ($p \in \Sigma$)

Core XPath

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]
`self::`*p* , pexpr, `not`(nexpr), nexpr `or` nexpr

Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

$$\mathsf{s} ::= \downarrow, \uparrow, \leftarrow, \rightarrow$$
$$\mathsf{a} ::= \mathsf{s} \mid \mathsf{s}^{+}$$
$$\mathsf{pexpr} ::= \mathsf{a} \mid \cdot \mid \mathsf{pexpr}/\mathsf{pexpr} \mid \mathsf{pexpr} \cup \mathsf{pexpr} \mid \mathsf{pexpr}[\mathsf{nexpr}]$$
$$\mathsf{nexpr} ::= p \mid \langle \mathsf{pexpr} \rangle \mid \neg \mathsf{nexpr} \mid \mathsf{nexpr} \vee \mathsf{nexpr} \qquad (p \in \Sigma)$$

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

Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

$$\mathsf{s} ::= \downarrow, \uparrow, \leftarrow, \rightarrow$$
$$\mathsf{a} ::= \mathsf{s} \mid \mathsf{s}^+$$
$$\text{pexpr} ::= \mathsf{a} \mid \cdot \mid \text{pexpr}/\text{pexpr} \mid \text{pexpr} \cup \text{pexpr} \mid \text{pexpr}[\text{nexpr}]$$
$$\text{nexpr} ::= p \mid \langle\text{pexpr}\rangle \mid \neg\text{nexpr} \mid \text{nexpr} \vee \text{nexpr} \qquad (p \in \Sigma)$$

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

We also consider single axis fragments of CoreXPath—notation CoreXPath(a) for a fixed axis a

Semantics of CoreXPath I

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

Semantics of CoreXPath I

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

Semantics of CoreXPath I

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,

Semantics of CoreXPath I

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

Semantics of CoreXPath I

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 II

$$
\begin{array}{ll}
\text{pexpr}: & \text{pairs (context node, reachable node)—subsets of } N^2 \\
[\![\mathsf{s}]\!]^T & = R_{\mathsf{s}} \\
[\![\mathsf{s}^+]\!]^T & = \text{the transitive closure of } R_{\mathsf{s}} \\
[\![\cdot]\!]^T & = \text{the identity relation on } N \\
[\![A/B]\!]^T & = \text{composition of } [\![A]\!]^T \text{ and } [\![B]\!]^T \\
[\![A \cup B]\!]^T & = \text{union of } [\![A]\!]^T \text{ and } [\![B]\!]^T \\
[\![A[\phi]]\!]^T & = \{(n, m) \in [\![A]\!]^T \mid m \in [\![\phi]\!]^T\}
\end{array}
$$

$$
\begin{array}{ll}
& \text{nexpr}: \text{ subsets of } N \\
[\![p]\!]^T & = \{n \in N \mid n \in V(p)\} \\
[\![\phi \wedge \psi]\!]^T & = [\![\phi]\!]^T \cap [\![\psi]\!]^T \\
[\![\neg\phi]\!]^T & = N \setminus [\![\phi]\!]^T \\
[\![\langle A \rangle]\!]^T & = \text{domain of } [\![A]\!]^T = \{n \mid (n, m) \in [\![A]\!]^T\}
\end{array}
$$

Remember what we've seen yesterday?

A (slightly modified) diagram of Johan Van Benthem:

$\underline{\mathfrak{W}}$				$\underline{\mathfrak{W}}^2$
unary properties	$\longrightarrow$	*modes*	$\longrightarrow$	binary relations
of states	$\longleftarrow$	*projections*	$\longleftarrow$	between states
propositional operators				program operators
ML				DRA/TRA

Examples of modes:

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

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

Examples of projections:

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

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

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

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

Examples of modes:

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

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

Examples of projections:

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

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

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

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

NOTE THAT:

$$\langle R\rangle = \sim\sim R$$

$$= R/R^{\smile} \cap \cdot$$

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

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)
which are exactly idempotent semirings with antidomain operation $\sim$
(also known as *dynamic negation*)

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)
which are exactly idempotent semirings with antidomain operation $\sim$
(also known as *dynamic negation*)

Short Core XPath

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

Short Core XPath

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

Syntax of Short Core XPath:

$s ::= \downarrow, \uparrow, \leftarrow, \rightarrow$
$a ::= s \mid s^{+}$
$exp ::= \cdot \mid a \mid exp/exp \mid exp \cup exp$

Short Core XPath

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

Syntax of Short Core XPath:

$\mathsf{s} ::= \downarrow, \uparrow, \leftarrow, \rightarrow$
$\mathsf{a} ::= \mathsf{s} \mid \mathsf{s}^{+}$
$\mathrm{exp} ::= \cdot \mid \mathsf{a} \mid \mathrm{exp}/\mathrm{exp} \mid \mathrm{exp} \cup \mathrm{exp} \mid ?p \mid \sim\mathrm{exp} \qquad (p \in \Sigma)$

Short Core XPath

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

Syntax of Short Core XPath:

$s ::= \downarrow, \uparrow, \leftarrow, \rightarrow$
$a ::= s \mid s^{+}$
$exp ::= \cdot \mid a \mid exp/exp \mid exp \cup exp \mid ?p \mid \sim exp \qquad (p \in \Sigma)$

Definition of a single axis fragment remains the same

Semantics of Short Core XPath

$$
\begin{array}{ll}
\text{pexpr}: & \text{pairs (context node, reachable node)—subsets of } N^2 \\
[\![\mathsf{s}]\!]^T & = R_{\mathsf{s}} \\
[\![\mathsf{s}^+]\!]^T & = \text{the transitive closure of } R_{\mathsf{s}} \\
[\![\cdot]\!]^T & = \text{the identity relation on } N \\
[\![A/B]\!]^T & = \text{composition of } [\![A]\!]^T \text{ and } [\![B]\!]^T \\
[\![A \cup B]\!]^T & = \text{union of } [\![A]\!]^T \text{ and } [\![B]\!]^T \\
[\![A[\phi]]\!]^T & = \{(n,m) \in [\![A]\!]^T \mid m \in [\![\phi]\!]^T\} \\
 & \qquad \text{nexpr}: \ \text{subsets of } N \\
[\![p]\!]^T & = \{n \in N \mid n \in V(p)\} \\
[\![\phi \wedge \psi]\!]^T & = [\![\phi]\!]^T \cap [\![\psi]\!]^T \\
[\![\neg\phi]\!]^T & = N \setminus [\![\phi]\!]^T \\
[\![\langle A \rangle]\!]^T & = \text{domain of } [\![A]\!]^T = \{n \mid (n,m) \in [\![A]\!]^T\}
\end{array}
$$

Semantics of Short Core XPath

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

$$
\begin{aligned}
[\![\mathsf{s}]\!]^T &= R_{\mathsf{s}}\\
[\![\mathsf{s}^+]\!]^T &= \text{the transitive closure of } R_{\mathsf{s}}\\
[\![\cdot]\!]^T &= \text{the identity relation on } N\\
[\![A/B]\!]^T &= \text{composition of } [\![A]\!]^T \text{ and } [\![B]\!]^T\\
[\![A \cup B]\!]^T &= \text{union of } [\![A]\!]^T \text{ and } [\![B]\!]^T
\end{aligned}
$$

$$
\begin{aligned}
[\![?p]\!]^T &= \{(n,n) \in N^2 \mid n \in V(p)\}\\
[\![\sim A]\!]^T &= \{(n,n) \in N^2 \mid \forall m.(n,m) \notin [\![A]\!]^T\}
\end{aligned}
$$

Back-and-forth Between Core XPath and SCX

One direction is easy:
$[\![\sim A]\!]^T = [\![\cdot[\neg\langle A \rangle]]\!]^T$

Back-and-forth Between Core XPath and SCX

One direction is easy:
$[\![\sim A]\!]^T = [\![\cdot[\neg\langle A\rangle]]\!]^T$
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

Back-and-forth Between Core XPath and SCX

One direction is easy:

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

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

$$[\![A]\!]^T = [\![t(A)]\!]^T \qquad \text{for all } A \in \text{pexpr}$$

$$[\![\cdot[\phi]]\!]^T = [\![t(\phi)]\!]^T \qquad \text{for all } \phi \in \text{nexpr}$$

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 $[\![P]\!]^T = [\![Q]\!]^T$

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

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

Which expressions are equivalent?

Let's give it a try:

- is it true that

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

- 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
  $\cdot \equiv \uparrow/\downarrow$?
- 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
  $\cdot \equiv \uparrow/\downarrow$? :(
- fine, how about
  $\cdot \equiv \downarrow/\uparrow$
  and
  $\uparrow/\downarrow \equiv \leftarrow^{+} \cup \cdot \cup \rightarrow^{+}$? :(
- 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

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

- fine, how about

  $\cdot \equiv \downarrow/\uparrow$

  and

  $\uparrow/\downarrow \equiv \leftarrow^{+} \cup \cdot \cup \rightarrow^{+}$?

- 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
  $\cdot \equiv \uparrow/\downarrow$?
- fine, how about
  $\cdot \equiv \downarrow/\uparrow$
  and
  $\uparrow/\downarrow \equiv \leftarrow^{+} \cup \cdot \cup \rightarrow^{+}$?
- One last try: how about
  $\sim\sim\downarrow \equiv \downarrow/\uparrow$
  and
  $\uparrow/\downarrow \equiv \leftarrow^{+} \cup \sim\sim\uparrow \cup \rightarrow^{+}$?

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

How does a complete axiomatization help?

- Solves the soundness problem:
  if all your rules can be derived from the axioms, they are valid

How does a complete axiomatization help?

- Solves the soundness problem:
  if all your rules can be derived from the axioms, they are valid
- Solves the completeness problem:
  if you can handle all the axioms, you did not forget anything

How does a complete axiomatization help?

- Solves the soundness problem:
  if all your rules can be derived from the axioms, they are valid
- Solves the completeness problem:
  if you can handle all the axioms, you did not forget anything
- Hopefully, it should also yield better rewrite strategies

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 standarized 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 standarized 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:
  $P \equiv P$
  $P \equiv Q \quad \Longrightarrow \quad Q \equiv P$

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:

$$
\begin{array}{lcl}
P \equiv P & & \\
P \equiv Q & \Longrightarrow & Q \equiv P \\
P \equiv Q \ \& \ Q \equiv R & \Longrightarrow & P \equiv R \\
P \equiv Q & \Longrightarrow & R \equiv R'
\end{array}
$$

  (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:

$$
\begin{array}{lcl}
P \equiv P & & \\
P \equiv Q & \Longrightarrow & Q \equiv P \\
P \equiv Q \;\&\; Q \equiv R & \Longrightarrow & P \equiv R \\
P \equiv Q & \Longrightarrow & R \equiv R'
\end{array}
$$

  (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:
  $$
  \begin{array}{lcl}
  P \equiv P & & \\
  P \equiv Q & \Longrightarrow & Q \equiv P \\
  P \equiv Q \ \& \ Q \equiv R & \Longrightarrow & P \equiv R \\
  P \equiv Q & \Longrightarrow & R \equiv R'
  \end{array}
  $$
  (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 cntnd: But What Use Are They For Us?

Q1 cntnd: But What Use Are They For Us?

An orthodox axiomatization

$\equiv$

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

Q1 cntnd: But What Use Are They For Us?

An orthodox axiomatization

$\equiv$

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 cntd: Is XPath Querying Any Better Off, Then?

Q2 cntd: Is XPath Querying Any Better Off, Then?

Short Answer

Yes.

Q2 cntd: 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 cntd: 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 cntd: 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 cntd: 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 cntd: 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

ISAx1	$(A \cup B) \cup C$	$\equiv$	$A \cup (B \cup C)$
ISAx2	$A \cup B$	$\equiv$	$B \cup A$
ISAx3	$A \cup A$	$\equiv$	A
ISAx4	$A/(B/C)$	$\equiv$	$(A/B)/C$
ISAx5	$\cdot/A$	$\equiv$	A
	$A/\cdot$	$\equiv$	A
ISAx6	$A/(B \cup C)$	$\equiv$	$A/B \cup A/C$
	$(A \cup B)/C$	$\equiv$	$A/C \cup B/C$
ISAx7	$\bot$	$\subseteq$	A

Distributive lattices, Kleene algebras, Tarski's relation algebras: they all have idempotent semiring reducts.
Idempotency is the axiom ISAx3.
$\bot$ abbreviates $\cdot\,[\neg\langle\cdot\rangle]$

Basic Axioms II: Predicate Axioms

PrAx1	$A[\neg\langle B\rangle]/B$	$\equiv$	$\bot$
PrAx2	$A[\phi \vee \psi]$	$\equiv$	$A[\phi] \cup A[\psi]$
PrAx3	$(A/B)[\phi]$	$\equiv$	$A/B[\phi]$
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

NdAx1	ϕ	$\equiv$	$\neg(\neg\phi \vee \psi) \vee \neg(\neg\phi \vee \neg\psi)$
NdAx2	$\langle A \cup B \rangle$	$\equiv$	$\langle A \rangle \vee \langle B \rangle$
NdAx3	$\langle A/B \rangle$	$\equiv$	$\langle A[\langle B \rangle] \rangle$
NdAx4	$\langle \cdot [\phi] \rangle$	$\equiv$	ϕ

Note how little was needed to ensure booleanity!
(by Huntington's result from the 1930's)
And NdAx2–NdAx4 just mimick 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:

PrAx1	$A\,[\neg\langle B\rangle]\,/B$	$\equiv$	$\bot$
PrAx2	$A\,[\phi \vee \psi]$	$\equiv$	$A\,[\phi] \cup A\,[\psi]$
PrAx3	$(A/B)\,[\phi]$	$\equiv$	$A/B\,[\phi]$
PrAx4	$\cdot\,[\langle\cdot\rangle]$	$\equiv$	$\cdot$
NdAx1	ϕ	$\equiv$	$\neg(\neg\phi \vee \psi) \vee \neg(\neg\phi \vee \neg\psi)$
NdAx2	$\langle A \cup B\rangle$	$\equiv$	$\langle A\rangle \vee \langle B\rangle$
NdAx3	$\langle A/B\rangle$	$\equiv$	$\langle A\,[\langle B\rangle]\,\rangle$
NdAx4	$\langle\cdot\,[\phi]\,\rangle$	$\equiv$	ϕ

Axioms in one-sorted signature

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

$$
\begin{array}{lcl}
\sim A/A & \equiv & \bot \\
\sim\sim A/A & \equiv & A \\
\sim(A/B)/A & \equiv & (\sim(A/B)/A)/\sim B \\
\sim(A \cup B) & \equiv & \sim A/\sim B \\
\sim A \cup \sim B & \equiv & \sim\sim(\sim A \cup \sim B)
\end{array}
$$

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:

$$\text{LinAx1} \quad \mathsf{s}[\neg\phi] \quad \equiv \quad \cdot[\neg\langle \mathsf{s}[\phi]\rangle]/\mathsf{s} \quad \text{for } \mathsf{s} \in \{\rightarrow, \leftarrow, \uparrow\}$$

This forces functionality of the corresponding axis

Axioms for Transitive Axes

One for node expressions, one for path expressions:

TransAx1	$\langle s^+[\phi]\rangle$	$\equiv$	$\langle s^+[\phi \wedge \neg\langle s^+[\phi]\rangle]\rangle$
TransAx2	s^+	$\equiv$	$s^+ \cup s^+/s^+$

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

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:

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

A Few Words About Proofs

- First, rewrite node expressions to simple node expressions:

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

They are isomorphic variants of modal formulas

A Few Words About Proofs

- First, rewrite node expressions to simple node expressions:

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

  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

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

(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

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

  (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

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

(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

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

  (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

$$p \wedge q \equiv \bot$$

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

$$p \wedge q \equiv \bot$$

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

Starting from the Other End

Instead of beginning with single axes
and then trying to combine two or more

Starting from the Other End

Instead of beginning with single axes
and then trying to combine two or more

LET'S GO FOR THE WHOLE CORE XPATH!

Axiom For Axes Dependencies

$$\text{TreeAx1}\begin{cases} s^{+}/s \cup s \equiv s^{+} \\ s/s^{+} \cup s \equiv s^{+} \end{cases}$$

$$\text{TreeAx2}\quad s\,[\phi]\,/s^{\smile} \equiv \cdot\,[\langle s\,[\phi]\,\rangle]\ (\text{ for s distinct than } \uparrow)$$

$$\text{TreeAx3}\quad \uparrow[\phi]\,/\!\downarrow \equiv (\leftarrow^{+} \cup \rightarrow^{+} \cup \cdot)\,[\langle\uparrow[\phi]\,\rangle]$$

$$\text{TreeAx4}\begin{cases} \leftarrow^{+} \equiv \leftarrow^{+}[\langle\uparrow\rangle] \\ \rightarrow^{+} \equiv \rightarrow^{+}[\langle\uparrow\rangle] \end{cases}$$

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

$$\langle s[\neg\langle\cdot\rangle]\rangle \equiv \neg\langle\cdot\rangle$$
$$\langle s[\phi \vee \psi]\rangle \equiv \langle s[\phi]\rangle \vee \langle s[\psi]\rangle$$
$$\phi \leq \neg\langle s[\neg\langle s^{\smile}[\phi]\rangle]\rangle$$
$$\langle s[\neg\phi]\rangle \wedge \langle s[\phi]\rangle \equiv \neg\langle\cdot\rangle (\text{ for s distinct than } \uparrow)$$
$$\langle s[\phi]\rangle \vee \langle s[\langle s^{+}[\phi]\rangle]\rangle \equiv \langle s^{+}[\phi]\rangle$$
$$\neg\langle s[\phi]\rangle \wedge \langle s^{+}[\phi]\rangle \leq \langle s^{+}[\neg\phi \wedge \langle s[\phi]\rangle]\rangle$$
$$\langle s[\langle\cdot\rangle]\rangle \leq \langle s^{+}[\neg\langle s[\langle\cdot\rangle]\rangle]\rangle$$

TransAx1 for $\downarrow^{+}$ and $\rightarrow^{+}$

$$\langle\downarrow[\neg\langle\leftarrow\rangle \wedge \neg\langle\rightarrow^{*}[\phi]\rangle]\rangle \leq \neg\langle\downarrow[\phi]\rangle$$
$$\langle\downarrow[\phi]\rangle \leq \langle\downarrow[\neg\langle\leftarrow\rangle]\rangle \wedge \langle\downarrow[\neg\langle\rightarrow\rangle]\rangle$$
$$\neg\langle\uparrow\rangle \leq \neg\langle\leftarrow\rangle \wedge \neg\langle\rightarrow\rangle$$

A Nasty Trick

A Nasty Trick

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

A Nasty Trick

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

. . . of a sort—a non-orthodox one! 

A 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 $A \equiv B$.

A 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

Complexity Theorem

Theorem

- *Query equivalence of Core XPath($\rightarrow^+, \leftarrow^+$), Core XPath($\uparrow^+$), Core XPath(s) (for 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.

Proofs

…by reductions to complexity results for modal logics like K, K4, 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.

Conclusions

We have seen:

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

Conclusions

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

Conclusions

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

Conclusions

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 (for finite sibling-ordered trees . . .)
- Results for fragments of XPath stronger than CoreXPath 1.0

Both are discussed in Balder's M4M slides