Complete Axiomatizations for XPath Fragments

Balder ten Cate / Tadeusz Litak / Maarten Marx

Amsterdam & Santa Cruz / London / Amsterdam

May 20, 2008
1 Introduction
   - XPath
   - Its Navigational Core
   - Query Equivalence Problem

2 Complete Axiomatizations, Logic and Algebra
   - An Axiomatization: What Is It Good For?
   - Connection with Logic and Algebra
   - Birkhoff’s Calculus
   - Two Questions

3 Axioms For Single Axis Fragments of XPath
   - Basic Axioms
   - Axioms for Transitive Axes

4 Full Core XPath
   - Node Expressions
   - Path Expressions
XML and Semi-structured Data
XML and Semi-structured Data

I guess we can skip the general introduction . . .
Example Document

No XML talk can do without its own example document:
No XML talk can do without its own example document:

```xml
<conference-dinner date='19-May-2008'>
```

In a picture:
Example Document

No XML talk can do without its own example document:

```xml
<conference-dinner date='19-May-2008'>
  <participant> Andrea, I’ll skip the desert </participant>
</conference-dinner>
```
Example Document

No XML talk can do without its own example document:

```xml
<conference-dinner date='19-May-2008'>
  <participant> Andrea, I’ll skip the desert </participant>
  <organizer>
    Desert? These were <i> antipasti </i>
  </organizer>
</conference-dinner>
```
Example Document

No XML talk can do without its own example document:

```xml
<conference-dinner date='19-May-2008'>
    <participant> Andrea, I’ll skip the desert </participant>
    <organizer>
        Desert? These were <i> antipasti </i>
    </organizer>
    <participant> Anti <b> what? </b> </participant>
</conference-dinner>
```
Example Document

No XML talk can do without its own example document:

```xml
<conference-dinner date='19-May-2008'>
    <participant> Andrea, I’ll skip the desert </participant>
    <organizer>
        Desert? These were <i> antipasti </i>
    </organizer>
    <participant> Anti <b> what? </b> </participant>
</conference-dinner>
```

In a picture:
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): ± 30 pages.**

Specification of XPath 2.0 (W3C, Nov ’05): ± 90 pages.
Specification of XPath 3.0: ± 270 pages? (Balder’s extrapolation)
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 2.0 (W3C, Nov ’05): ± 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: `/node[@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 2.0 (W3C, Nov ’05): ± 90 pages.
- Specification of XPath 3.0: ± 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’02 and Gottlob, Koch and Pichler’02
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’02 and Gottlob, Koch and Pichler’02

More precisely speaking, an almost isomorphic variant . . .
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:

\[
\begin{align*}
s & ::= \downarrow, \uparrow, \leftarrow, \rightarrow \\
a & ::= s \mid s^+ \\
pexpr & ::= a \mid . \mid pexpr/pexpr \mid pexpr \cup pexpr \mid pexpr[nexpr]
\end{align*}
\]
Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

\[
\begin{align*}
\mathbf{s} & ::= \downarrow, \uparrow, \leftarrow, \rightarrow \\
\mathbf{a} & ::= \mathbf{s} \mid \mathbf{s}^+ \\
\mathbf{pexpr} & ::= \mathbf{a} \mid . \mid \mathbf{pexpr}/\mathbf{pexpr} \mid \mathbf{pexpr} \cup \mathbf{pexpr} \mid \mathbf{pexpr}[\mathbf{nexpr}] \\
\mathbf{nexpr} & ::= p \mid \langle \mathbf{pexpr} \rangle \mid \neg \mathbf{nexpr} \mid \mathbf{nexpr} \land \mathbf{nexpr} \quad (p \in \Sigma)
\end{align*}
\]
Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

- \( s ::= \downarrow, \uparrow, \leftarrow, \rightarrow \)
- \( a ::= s \mid s^+ \)
- \( pexpr ::= a \mid . \mid pexpr/pexpr \mid pexpr \cup pexpr \mid pexpr[nexpr] \)
- \( nexpr ::= p \mid \langle pexpr \rangle \mid \neg nexpr \mid nexpr \land nexpr \quad (p \in \Sigma) \)

We also consider **single axis fragments** of CoreXPath—notation
\( \text{CoreXPath}(a) \) for a fixed axis \( a \)
Our notation is a bit different from the official XPath notation.
Our notation is a bit different from the official XPath notation.

next-right-sibling and next-left-sibling are our additions:
in XPath 1.0 those axes have to be simulated using positional predicates
Our notation is a bit different from the official XPath notation.

- `next-right-sibling` and `next-left-sibling` are our additions:
in XPath 1.0 those axes have to be simulated using positional predicates
(I am not happy about it)
Our notation is a bit different from the official XPath notation.

- next-right-sibling and next-left-sibling are our additions: in XPath 1.0 those axes have to be simulated using positional predicates (I am not happy about it)
- They are unproblematic in XPath 2.0, though
Our notation is a bit different from the official XPath notation.

- `next-right-sibling` and `next-left-sibling` are our additions:
  in XPath 1.0 those axes have to be simulated using positional predicates
  (I am not happy about it)
  They are unproblematic in XPath 2.0, though

- Finally, `following` and `preceding` axes are also missing—but these are definable
With such a simple syntax, our data model can be also simplified:

- XML documents are now represented as finite sibling-ordered node labelled trees.
XML Data Model Revisited

With such a simple syntax, our data model can be also simplified:

- XML documents are now represented as finite sibling-ordered node labelled trees.
- So, an XML document is a tuple $T = (N, R_{\downarrow}, R_{\rightarrow}, V)$ where
XML Data Model Revisited

With such a simple syntax, our data model can be also simplified:

- XML documents are now represented as finite sibling-ordered node labelled trees.
- So, an XML document is a tuple $T = (N, R_\downarrow, R_\rightarrow, V)$ where
  - $N$ is the set of nodes,
XML Data Model Revisited

With such a simple syntax, our data model can be also simplified:

- XML documents are now represented as finite sibling-ordered node labelled trees.
- So, an XML document is a tuple $T = (N, R_{\downarrow}, R_{\rightarrow}, V)$ where
  - $N$ is the set of nodes,
  - $R_{\downarrow}$ and $R_{\rightarrow}$ are the child and next-right-sibling relations, and
XML Data Model Revisited

With such a simple syntax, our data model can be also simplified:

- XML documents are now represented as finite sibling-ordered node labelled trees.
- So, an XML document is a tuple $T = (N, R↓, R→, V)$ where
  - $N$ is the set of nodes,
  - $R↓$ and $R→$ are the child and next-right-sibling relations, and
  - $V : N \rightarrow \Sigma$. 
pexpr : pairs (context node, reachable node)—subsets of $N^2$

$$[s]^M = R_s$$

$$[s^+]^M = \text{the transitive closure of } R_s$$

$$[.]^M = \text{the identity relation on } N$$


$$[A \cup B]^M = \text{union of } [A]^M \text{ and } [B]^M$$

$$[A[\phi]]^M = \{(n, m) \in [A]^M | m \in [\phi]^M\}$$

nexpr : subsets of $N$

$$[p]^M = \{n \in N | V(n) = p\}$$

$$[\phi \land \psi]^M = [\phi]^M \cap [\psi]^M$$

$$[\neg \phi]^M = N \setminus [\phi]^M$$

$$[\langle A \rangle]^M = \text{domain of } [A]^M = \{n | (n, m) \in [A]^M\}$$
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 $[P]^M = [Q]^M$
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.

One of reasons why we consider Core XPath only:

the whole XPath would be too big to allow a complete axiomatization
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:

<table>
<thead>
<tr>
<th>logic:</th>
<th>algebras:</th>
<th>databases:</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulas</td>
<td>terms</td>
<td>query plans</td>
</tr>
</tbody>
</table>
Logicians and algebraists have long studied similar problems in a different disguise:

<table>
<thead>
<tr>
<th>logic:</th>
<th>algebras:</th>
<th>databases:</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulas</td>
<td>terms</td>
<td>query plans</td>
</tr>
<tr>
<td>tautologies</td>
<td>equations</td>
<td>query equivalences</td>
</tr>
</tbody>
</table>
Logic—Algebra—Query Languages

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

<table>
<thead>
<tr>
<th>logic:</th>
<th>algebras:</th>
<th>databases:</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulas</td>
<td>terms</td>
<td>query plans</td>
</tr>
<tr>
<td>tautologies</td>
<td>equations</td>
<td>query equivalences</td>
</tr>
<tr>
<td>inference rules</td>
<td></td>
<td>rewrite rules</td>
</tr>
</tbody>
</table>
Logic—Algebra—Query Languages

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

<table>
<thead>
<tr>
<th>logic:</th>
<th>algebras:</th>
<th>databases:</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulas</td>
<td>terms</td>
<td>query plans</td>
</tr>
<tr>
<td>tautologies</td>
<td>equations</td>
<td>query equivalences</td>
</tr>
<tr>
<td>inference rules</td>
<td>rewrite rules</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>logic</th>
<th>algebras</th>
<th>databases</th>
</tr>
</thead>
<tbody>
<tr>
<td>formulas</td>
<td>terms</td>
<td>query plans</td>
</tr>
<tr>
<td>tautologies</td>
<td>equations</td>
<td>query equivalences</td>
</tr>
<tr>
<td>inference rules</td>
<td>rewrite rules</td>
<td></td>
</tr>
</tbody>
</table>

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

- $P \equiv P$

- $P \equiv Q \equiv P$
Birkhoff’s Calculus For Equational Logic

Definition

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

\[
\begin{align*}
P & \equiv P \\
P & \equiv Q & \implies & \quad Q \equiv P
\end{align*}
\]

An axiomatization using Birkhoff’s rules only is **orthodox**. Clearly, these rules preserve validity.
**Definition**

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

\[
\begin{align*}
P & \equiv P \\
P & \equiv Q & \implies & Q \equiv P \\
P & \equiv Q \land Q \equiv R & \implies & P \equiv R
\end{align*}
\]
Birkhoff’s Calculus For Equational Logic

Definition

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

\[
\begin{align*}
    P \equiv P \\
    P \equiv Q & \implies Q \equiv P \\
    P \equiv Q & \land Q \equiv R \implies P \equiv R \\
    P \equiv Q & \implies R \equiv R' \\
\end{align*}
\]

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

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

- $P \equiv P$
- $P \equiv Q$ \implies Q \equiv P$
- $P \equiv Q \land Q \equiv R$ \implies P \equiv R$
- $P \equiv Q$ \implies 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 $\Gamma$ be a set of equivalences.
- Equivalence $P \equiv Q$ is derivable from $\Gamma$ if it can be obtained by the following rules:
  - $P \equiv P$
  - $P \equiv Q \quad \implies \quad Q \equiv P$
  - $P \equiv Q \land Q \equiv R \quad \implies \quad P \equiv R$
  - $P \equiv Q \quad \implies \quad R \equiv R'$
  ($R'$ 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?

What is so great about this derivation system? Is it . . .
Q1: Why Birkhoff Calculus?

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

- . . . the definition itself?
Q1: Why Birkhoff Calculus?

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?

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?

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 cont'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)
Q1 continued: 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 cont'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/relational algebras?*

After all, there has been nothing XML specific to what I said . . .
Q2: Anything Special about XPath?

Question

*How about complete axiomatizations for SQL/relational algebras?*

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

Answer

*Even with no more than three attributes, you soon run into un axiomatizability 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/relational algebras?

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 contd: Is Core XPath Any Better Off, Then?

Yes.

Two reasons:

Precisely because we can isolate the navigational core (would not make much sense in the relational context)

Core XPath is related to well-behaved formalisms:

- Node expressions— to modal logic
- Path expressions—to arrow logics
- and (reducts of) Tarski's relation algebras

The duality of path and node expressions:

resembles the logic of programs (PDL), logics used in verification like LTL and CTL...

We will make full use of these correspondences in what follows.
Q2 cntd: Is Core XPath Any Better Off, Then?

Answer

Yes.
Q2 cntd: Is Core XPath Any Better Off, Then?

Answer

Yes.

Two reasons:

- Precisely because we can isolate the navigational core (would not make much sense in the relational context)
Q2 cntd: Is Core XPath Any Better Off, Then?

Answer

Yes.

Two reasons:

- Precisely because we can isolate the navigational core (would not make much sense in the relational context)
- Core XPath is related to well-behaved formalisms:
  - Node expressions—to modal logic
Q2 cntd: Is Core XPath Any Better Off, Then?

Answer

Yes.

Two reasons:

- Precisely because we can isolate the navigational core (would not make much sense in the relational context)
- Core XPath is related to well-behaved formalisms:
  - Node expressions—to modal logic
  - Path expressions—to arrow logics and (reducts of) Tarski's relation algebras
Q2 cntd: Is Core XPath Any Better Off, Then?

Answer

Yes.

Two reasons:

- Precisely because we can isolate the navigational core (would not make much sense in the relational context)
- Core XPath is related to well-behaved formalisms:
  - Node expressions—to modal logic
  - Path expressions—to arrow logics and (reducts of) Tarski’s relation algebras
  - The duality of path and node expressions: resembles the logic of programs (PDL), logics used in verification like LTL and CTL . . .
Q2 cntd: Is Core XPath Any Better Off, Then?

Answer
Yes.

Two reasons:

- Precisely because we can isolate the navigational core (would not make much sense in the relational context)
- Core XPath is related to well-behaved formalisms:
  - Node expressions—to modal logic
  - Path expressions—to arrow logics and (reducts of) Tarski’s relation algebras
  - The duality of path and node expressions: resembles the logic of programs (PDL), logics used in verification like LTL and CTL . . .

We will make full use of these correspondences in what follows.
Basic Axioms I: Idempotent Semirings

\[
\begin{align*}
\text{ISAx1} & \quad (A \cup B) \cup C \equiv A \cup (B \cup C) \\
\text{ISAx2} & \quad A \cup B \equiv B \cup A \\
\text{ISAx3} & \quad A \cup A \equiv A \\
\text{ISAx4} & \quad A/(B/C) \equiv (A/B)/C \\
\text{ISAx5} & \quad \begin{array}{l}
. / A \equiv A \\
A/ . \equiv A \\
A/(B \cup C) \equiv A/B \cup A/C \\
(A \cup B)/C \equiv A/C \cup B/C \\
\perp \subseteq A
\end{array}
\end{align*}
\]

Distributive lattices, Kleene algebras, Tarski’s relation algebras: they all have \textbf{idempotent semiring} reducts. Idempotency is the axiom ISAx3. \( \perp \) abbreviates \( \lnot \langle . \rangle \).
Basic Axioms II: Predicate Axioms

\begin{align*}
\text{PrAx1} & \quad A[\neg \langle B \rangle]/B \equiv \bot \\
\text{PrAx2} & \quad A[\phi \lor \psi] \equiv A[\phi] \cup A[\psi] \\
\text{PrAx3} & \quad (A/B)[\phi] \equiv A/B[\phi] \\
\text{PrAx4} & \quad \langle . \rangle \equiv .
\end{align*}

In Tarski’s relation algebras and XPath 2.0, predicates can be defined away.
Basic Axioms III: Node Axioms

\( \text{NdAx1} \quad \varphi \quad \equiv \quad \neg (\neg \varphi \lor \psi) \lor \neg (\neg \varphi \lor \neg \psi) \)

\( \text{NdAx2} \quad \langle A \cup B \rangle \quad \equiv \quad \langle A \rangle \lor \langle B \rangle \)

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

\( \text{NdAx4} \quad \langle . [\varphi] \rangle \quad \equiv \quad \varphi \)

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)
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(↓), the child-only fragment!

**Theorem**

*The axioms presented so far are complete for all valid equivalences of Core XPath(↓).*
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(↓), the child-only fragment!

**Theorem**

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

In order to find more interesting equivalences, we have to move to other fragments
Axioms for Transitive Axes

One for node expressions, one for path expressions:

\[
\text{TransAx1} \quad \langle s^+ [\phi] \rangle \equiv \langle s^+ [\phi \land \neg \langle s^+ [\phi] \rangle] \rangle \\
\text{TransAx2} \quad s^+ \equiv s^+ \cup s^+ / s^+
\]

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.
Completeness for Descendant-only Fragment

Theorem

All the axioms presented up to now are complete for Core XPath(↓⁺)
Completeness for Descendant-only Fragment

**Theorem**

*All the axioms presented up to now are complete for Core XPath\((\downarrow^+)\)*

That is, for queries containing only the descendant axis it is enough to add

\[
\langle \downarrow^+ [\phi] \rangle \equiv \langle \downarrow^+ [\phi \land \neg \langle \downarrow^+ [\phi] \rangle] \rangle \\
\downarrow^+ \equiv \downarrow^+ \cup \downarrow^+/\downarrow^+
\]

to the basic axioms to axiomatize all valid equivalences.
A Few Words About The Proof

- First, rewrite node expressions to simple node expressions:

\[
siNode ::= \langle . \rangle \mid p \mid \langle \downarrow^+ [siNode] \rangle \mid \neg siNode \mid siNode \lor siNode
\]
A Few Words About The Proof

- First, rewrite node expressions to simple node expressions:

  \[ \text{siNode} ::= \langle . \rangle \mid p \mid \langle \downarrow^+ \text{[siNode]} \rangle \mid \neg \text{siNode} \mid \text{siNode} \lor \text{siNode} \]

  They are isomorphic variants of **basic modal formulas**
A Few Words About The Proof

- First, rewrite node expressions to simple node expressions:

\[ \text{siNode ::= } \langle \cdot \rangle \mid p \mid \langle \downarrow^+ \text{[siNode]} \rangle \mid \neg \text{siNode} \mid \text{siNode} \lor \text{siNode} \]

They are isomorphic variants of basic modal formulas.

- Using normal form theorems for modal logic, we provide a completeness proof for node expressions.
A Few Words About The Proof cntd.

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

\[ S = .[β_1]/↓+ [β_2]/.../↓+ [β_ℓ], \]

(all \( β_i \) are normal forms of the same nesting degree)
A Few Words About The Proof cntd.

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

\[ S = . [\beta_1] / \downarrow^+ [\beta_2] / \ldots / \downarrow^+ [\beta_\ell], \]

(all \( \beta_i \) are normal forms of the same nesting degree)

- We prove that for every two such expressions either
A Few Words About The Proof cntd.

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

\[ S = .[\beta_1]/\downarrow^+ [\beta_2]/\ldots/\downarrow^+[\beta_\ell], \]

(all \( \beta_i \) are normal forms of the same nesting degree)

We prove that for every two such expressions either

- one is a subsequence of the other—provably contained or
A Few Words About The Proof cntd.

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

\[ S = .[\beta_1]/\downarrow^+[\beta_2]/\ldots/\downarrow^+[\beta_\ell], \]

(all \( \beta_i \) are normal forms of the same nesting degree)

- 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: did I cheat a little?

There is a fact about XML trees we did not take into account: The labels are disjoint! However, this is easy to fix: add $p \land q \equiv \bot$ for $p \neq q$.

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.

Another option: think of labels as modelling not only tag names, but also attribute-value pairs.
Aside: did I cheat a little?

There is a fact about XML trees we did not take into account
Aside: did I cheat a little?

There is a fact about XML trees we did not take into account

The labels are disjoint!
Aside: did I cheat a little?

There is a fact about XML trees we did not take into account

The labels are disjoint!

However, this is easy to fix: add

\[ p \land q \equiv \perp \quad \text{for } p \neq q \]

This axiom itself is not substitution-invariant, this is why we do not like it
Aside: did I cheat a little?

There is a fact about XML trees we did not take into account

The labels are disjoint!

However, this is easy to fix: add

\[ p \land q \equiv \bot \quad \text{for } p \neq q \]

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
Aside: did I cheat a little?

There is a fact about XML trees we did not take into account

The labels are disjoint!

However, this is easy to fix: add

\[ p \land q \equiv \bot \quad \text{for } p \neq q \]

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.
Another option: think of labels as modelling not only tag names, but also attribute-value pairs.
Okay, What Next?

There is some disagreement between us now: whether we should go for other single axis fragments.
Okay, What Next?

There is some disagreement between us now: whether we should go for other single axis fragments (candidates:
\[ s[\phi] \equiv . \neg s[\neg\phi] \] /s
and
\[ \langle s^+ [\phi] \rangle / s^+ \equiv s^+ [\phi] \cup s^+ [\phi] / s^+ \cup s^+ [\langle s^+ [\phi] \rangle] \]
Okay, What Next?

There is some disagreement between us now: whether we should go for other single axis fragments (candidates:

\( s[\phi] \equiv .[\neg<s[\neg\phi]]/s \)

and

\( <s^+[\phi]/s^+ \equiv s^+[\phi] \cup s^+[\phi]/s^+ \cup s^+[<s^+[\phi]] > \)

or try to axiomatize a fragment with two axes: descendant together with following-or-descendant
Or Just Go For The Big Fish?

For the time being, let us start from the other end:
Or Just Go For The Big Fish?

For the time being, let us start from the other end:
instead of beginning with single axes
and then trying to combine two or more …
Or Just Go For The Big Fish?

For the time being, let us start from the other end:

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

. . . go for the whole CoreXPath 1.0
Axiom For Axes Dependencies

\[
\begin{align*}
\text{TreeAx1} & \quad s^+/s \cup s \equiv s^+ \\
\text{TreeAx2} & \quad s/s^+ \cup s \equiv s^+ \\
\text{TreeAx3} & \quad s[\phi]/s^{-1} \equiv .[s[\phi]] \text{ (for } s \text{ distinct than } \uparrow) \\
\text{TreeAx4} & \quad \uparrow[\phi]/\downarrow \equiv (\leftarrow^+ \cup \rightarrow^+ \cup .)[\langle \uparrow \rangle] \\
\end{align*}
\]

TreeAx1 says: $s^+$ is a transitive closure of $s$

TreeAx2 says non-child steps 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’95
\[ \langle s [\neg \langle . \rangle] \rangle \equiv \neg \langle . \rangle \]
\[ \langle s [\phi \lor \psi] \rangle \equiv \langle s [\phi] \rangle \lor \langle s [\psi] \rangle \]
\[ \phi \leq \langle s [\neg \langle s^{-1} [\phi] \rangle] \rangle \]
\[ \langle s [\phi] \rangle \lor \langle s [s^+ [\phi]] \rangle \equiv \langle s^+ [\phi] \rangle \]
\[ \neg \langle s [\phi] \rangle \land \langle s^+ [\phi] \rangle \leq \langle s^+ [\neg \phi \land \langle s [\phi] \rangle] \rangle \]
\[ \langle s [\langle . \rangle] \rangle \]
\[ \langle \downarrow [\neg \langle \leftarrow \rangle \land \neg \langle \rightarrow^* [\phi] \rangle] \rangle \leq \langle \downarrow [\neg [\phi]] \rangle \]
\[ \langle \downarrow [\phi] \rangle \leq \langle \downarrow [\neg \langle \leftarrow \rangle] \rangle \land \langle \downarrow [\neg \langle \rightarrow \rangle] \rangle \]
\[ \neg \langle \uparrow \rangle \leq \neg \langle \leftarrow \rangle \land \neg \langle \rightarrow \rangle \]

LOFT axioms include also boolean tautologies and TransAx1 for $\downarrow^+$ and $\rightarrow^+$. 

Balder ten Cate / Tadeusz Litak / Maarten Marx

Complete Axiomatizations for XPath Fragments (34/37)
A Nasty Trick

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

(Sep)

IF \( \langle A[p] \rangle \equiv \langle B[p] \rangle \) AND \( p \) does not occur in \( A \) and \( B \) THEN \( A \equiv B \).

It does not sit too well with the labelling axiom either.
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! 😊
We can use this to provide an axiomatization for path expressions ... 

... of a sort—a non-orthodox one! 😞

Add a separability rule:

\[(\text{Sep}) \quad \text{IF } \langle A[p] \rangle \equiv \langle B[p] \rangle \]

\[\text{AND } p \text{ does not occur in } A \text{ and } B \]

\[\text{THEN } A \equiv B.\]
A Nasty Trick

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

. . . of a sort—a non-orthodox one! 😝

Add a separability rule:

\[(\text{Sep}) \text{ IF } \langle A[p] \rangle \equiv \langle B[p] \rangle \]

AND \( p \) does not occur in \( A \) and \( B \)

THEN \( A \equiv B \).

It does not sit too well with the labelling axiom either . . .
Summary

We have used logical and algebraic techniques to axiomatize query equivalence in fragments of Core XPath.
Summary

We have used logical and algebraic techniques to axiomatize query equivalence in fragments of Core XPath.

We saw nice results for

- child-only fragment $\text{CoreXPath}('^')$

- descendant-only fragment $\text{CoreXPath}('\downarrow')$

- node expressions of the full CoreXPath.
We have used logical and algebraic techniques to axiomatize query equivalence in fragments of Core XPath. We saw nice results for

- child-only fragment $\text{CoreXPath}(\downarrow)$
- descendant-only fragment $\text{CoreXPath}(\downarrow^+)$
We have used logical and algebraic techniques to axiomatize query equivalence in fragments of Core XPath. We saw nice results for

- child-only fragment $\text{CoreXPath}(\downarrow)$
- descendant-only fragment $\text{CoreXPath}(\downarrow^+)$
- node expressions of the full CoreXPath
Summary

We have used logical and algebraic techniques to axiomatize query equivalence in fragments of Core XPath. We saw nice results for

- child-only fragment CoreXPath(↓)
- descendant-only fragment CoreXPath(↓⁺)
- node expressions of the full CoreXPath

We also saw a (rather ad hoc) axiomatization of the whole language.
Future Work and Open Problems

- Does the full Core XPath 1.0 admit axiomatization without non-orthodox rules?
Future Work and Open Problems

- Does the full Core XPath 1.0 admit axiomatization without non-orthodox rules?
- Prove conjectures for remaining single axis fragments
Future Work and Open Problems

- Does the full Core XPath 1.0 admit axiomatization without non-orthodox rules?
- Prove conjectures for remaining single axis fragments
- Axiomatize restricted, e.g., positive fragments of XPath properly contained between the fragment of Benedikt, Fan and Kuper’05 and Core XPath 1.0
Future Work and Open Problems

- Does the full Core XPath 1.0 admit axiomatization without non-orthodox rules?
- Prove conjectures for remaining single axis fragments
- Axiomatize restricted, e.g., positive fragments of XPath properly contained between the fragment of Benedikt, Fan and Kuper’05 and Core XPath 1.0

Challenge

Develop more efficient rewrite strategies based on these axiomatizations