XPath from a Logical Point of View

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

Department of Computer Science
University of Leicester

15 December 2010
MGS Christmas Dinner Apéritif
Expressive power:
Marx and De Rijke.
Semantic characterizations of navigational XPath.
SIGMOD Record 34(2), 2005

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

Ten Cate and Segoufin.
XPath, transitive closure logic, and nested tree walking automata.

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

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

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

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

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

Ten Cate and Lutz.
Query containment in very expressive XPath dialects.
PODS'07.
Expressive power:

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


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

Axiomatization:


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

Complexity:


Ten Cate and Lutz. *Query containment in very expressive XPath dialects*. PODS’07.
What This Talk Is About

Abstract of Abstract

I will sketch how to use an intimate relationship between

• subsets and extensions of XPath 1.0 and 2.0 (in particular for their "navigational core") and

• well-understood logical and algebraic formalisms
to derive results on

• complete axiomatizations
• computational complexity and
• expressive power
for XPath fragments
Abstract of Abstract
I will sketch how to use an intimate relationship between
• subsets and extensions of XPath 1.0 and 2.0 (in particular for their "navigational core") and
• well-understood logical and algebraic formalisms
to derive results on
• complete axiomatizations
• computational complexity and
• expressive power
for XPath fragments
XML and Semi-structured Data
XML and Semi-structured Data

XML

eXtensible Markup Language
XML 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

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

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

(no DTD given, but you can easily come up with one)
What we’ll see through our dim glasses
What we’ll see through our dim glasses

Either this …

(we cannot even see attributes, each node is labelled with a single label: its name)
What we’ll see through our dim glasses

or that ...

(attribute-value pairs are additional labels)
What we’ll see through our dim glasses

or perhaps ...

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

see Ranko Lazić’s talk last week for a more sophisticated approach to trees with values
What we’ll see through our dim glasses

or perhaps ...

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

At any rate, we are too blind to see actual text content
XPath 1.0: W3C Specification

- Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer
XPath 1.0: W3C Specification

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

- Primary purpose: *to address parts of an XML document*
XPath 1.0: W3C Specification

- Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer
- **Primary purpose:** to address parts of an XML document
- **In support of this primary purpose,** it also provides basic facilities for manipulation of strings, numbers and booleans
XPath 1.0: W3C Specification

- Provides a common syntax and semantics for functionality shared between [XQuery], XSL Transformations and XPointer
- **Primary purpose:** to address parts of an XML document
- **In support of this primary purpose,** it also provides basic facilities for manipulation of strings, numbers and booleans
- **Uses a compact, non-XML syntax** to facilitate use of XPath within URIs and XML attribute values
XPath 1.0: W3C Specification

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

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

XPath 1.0 specification (W3C, Nov '99): 37 pages.
XPath 2.0 specification (W3C, Jan '07): 122 pages.
XPath 3.0: ...?
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//*`
- **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[string(body,1,3)="It’s"]`
- **Arithmetical functions.** . . .
-  . . .
- **XPath 1.0 specification (W3C, Nov ’99): 37 pages.**
Samples of XPath Expressions

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

 XPath 2.0 specification (W3C, Jan ’07): 122 pages.
Samples of XPath Expressions

- **Unions.** For example: `/note/from | /note/to`.
- **Counting.** For example: `/node/to[1]`
- **Descendant and ancestor steps.** For example: `/node//*`
- **Filters.** For example: `/note[from]/to`
- **Attributes.** For example: `/note[@date="10-nov-2006"]`
- **String functions.** For example: 
  `/note[substring(body,1,3)="It’s"]`
- **Arithmetical functions.** . . .
- . . .
- XPath 2.0 specification (W3C, Jan ’07): 122 pages.
- XPath 3.0: . . .?
One of the pleasures of XPath 1.0 (like XML itself) was the brevity of the specification: a mere 30 pages. I haven’t tried to print the XPath 2.0 specification, but it is certainly vastly longer. What’s more, it is split between multiple documents. The main language specification at http://www.w3.org/TR/xpath20 points to subsidiary documents describing the data model, the function library, and the formal semantics, all at considerable length. As with a comparison between the US Constitution and the proposed EU Constitution, the length of the document tells us more about the number of people involved in defining it than about the benefits it offers. I would estimate that in reality the 2.0 language is about twice the size of XPath 1.0. (...) But whether the increased word count in the spec adds precision and clarity, or merely creates opportunities for errors and inconsistencies to creep in, is anyone’s guess.
Core XPath 1.0

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

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

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:

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

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

Restricting the set of axes is quite common: recall James Cheney’s or Ranko Lazi´c’s talks last week for fresh examples.
Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

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

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

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

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

Core XPath has two types of expressions:

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

Syntax of Core XPath:

\[
\begin{align*}
  s & ::= \downarrow, \uparrow, \Leftarrow, \Rightarrow \\
  a & ::= s | s^+ \\
  pexpr & ::= a | \cdot | pexpr/pexpr | pexpr \cup pexpr | pexpr[nexpr] \\
  nexpr & ::= p | \langle pexpr \rangle | \neg nexpr | nexpr \lor nexpr \quad (p \in \Sigma)
\end{align*}
\]
Core XPath

Core XPath has two types of expressions:

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

Syntax of Core XPath:

```
children::* | parent::* | preceding-sibling::*[1] | following-sibling::*[1]
```

```
a ::= s | s^+
pexpr ::= a | · | pexpr/pexpr | pexpr ∪ pexpr | pexpr[nexpr]
nexpr ::= p | ⟨pexpr⟩ | ¬nexpr | nexpr ∨ nexpr (p ∈ Σ)
```
Core XPath has two types of expressions:

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

Syntax of Core XPath:

```
children::*
parent::*
preceding-sibling::*[1]
following-sibling::*[1]
... descendant::*
ancestor::*
preceding-sibling::*
following-sibling::*
descendant::*
ancestor::*
preceding-sibling::*
following-sibling::*
```

```
pexpr ::= a | · | pexpr/pexpr | pexpr ∪ pexpr | pexpr[nexpr]
nexpr ::= p | ⟨pexpr⟩ | ¬nexpr | nexpr ∨ nexpr (p ∈ Σ)
```

(Our notation is a bit different from the official XPath notation)
We also consider single and restricted axis fragments of CoreXPath—notation CoreXPath(a) for a single axis and CoreXPath(A) for a set of axes Restricting the set of axes is quite common: recall James Cheney’s or Ranko Lazić’s talks last week for fresh examples
Core XPath has two types of expressions:
- Path expressions define binary relations
- Node expressions define sets of nodes

Syntax of Core XPath:

```
children::* , parent::* , preceding-sibling::*[1], following-sibling::*[1]
... descendant::* , ancestor::* , preceding-sibling::* , following-sibling::*
... self::* , pexpr/pexpr , pexpr | pexpr , pexpr[nexpr]

nexpr ::= p | ⟨pexpr⟩ | ¬nexpr | nexpr ∨ nexpr (p ∈ Σ)
```
Core XPath

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

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

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

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

Core XPath has two types of expressions:

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

Syntax of Core XPath:

\[
s ::= \downarrow, \uparrow, \leftarrow, \Rightarrow
\]

\[
a ::= s \mid s^+\]

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

\[
nexpr ::= p \mid \langle pexpr \rangle \mid \neg nexpr \mid nexpr \lor nexpr \quad (p \in \Sigma)
\]

(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:

\[
\begin{align*}
s & ::= \downarrow, \uparrow, \leftarrow, \rightarrow \\
a & ::= s | s^+ \\
pexpr & ::= a | \cdot | pexpr/pexpr | pexpr \cup pexpr | pexpr[nexpr] \\
nexpr & ::= p | \langle pexpr \rangle | \neg nexpr | nexpr \lor nexpr \quad (p \in \Sigma)
\end{align*}
\]

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

We also consider single and restricted axis fragments of CoreXPath—notation \text{CoreXPath}(a) for a single axis and \text{CoreXPath}(A) for a set of axes

Restricting the set of axes is quite common: recall James Cheney’s or Ranko Lazić’s talks last week for fresh examples
As said above, we see XML documents as finite sibling-ordered node labelled trees: ideal abstraction for such a simple syntax
As said above, we see XML documents as finite sibling-ordered node labelled trees: ideal abstraction for such a simple syntax

**XML document**
A tuple $T = (N, R_{\downarrow}, R_{\rightarrow}, V)$ where
As said above, we see XML documents as finite sibling-ordered node labelled trees: ideal abstraction for such a simple syntax

**XML document**

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

- \( N \) is the set of nodes,
As said above, we see XML documents as finite sibling-ordered node labelled trees: ideal abstraction for such a simple syntax

**XML document**

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

- $N$ is the set of nodes,
- $R_{\downarrow}$ and $R_{\Rightarrow}$ are ‘child’ and ‘next sibling’ relations of a finite tree, and
Semantics of CoreXPath

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

**XML document**

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

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

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

$[s]^T = R_s$
$[s^+]^T =$ the transitive closure of $R_s$
$[\cdot]^T =$ the identity relation on $N$
$[A[\phi]]^T = \{(n, m) \in [A]^T \mid m \in [\phi]^T\}$

nexpr : subsets of $N$

$[\rho]^T = \{n \in N \mid n \in V(\rho)\}$
$[\phi \land \psi]^T = [\phi]^T \cap [\psi]^T$
$[\neg \phi]^T = N \setminus [\phi]^T$
$[\langle A \rangle]^T =$ domain of $[A]^T = \{n \mid (n, m) \in [A]^T\}$
A (slightly modified) diagram of Johan Van Benthem

\[ W \]

unary properties \[ \rightarrow \] modes \[ \rightarrow \] binary relations

of states \[ \leftarrow \] projections \[ \leftarrow \] between states

propositional operators

ML

DRA/TRA
Examples of modes:

\[ ?X := \{ \langle x, x \rangle \mid x \in X \} \]  
(testing)

\[ !X := \{ \langle w, x \rangle \mid w \in Y, x \in X \} \]  
(realizing)

Examples of projections:

\[ \langle R \rangle := \{ w \in Y \mid \exists v \in Y . wR^Y v \} \]  
(domain)

\[ \pi^{-1}(R) := \{ w \in Y \mid \exists v \in Y . vR^Y w \} \]  
(codomain)

\[ \sim R := \{ w \in Y \mid \forall v \in Y . \neg (wR^Y v) \} \]  
(antidomain)

\[ \Delta(R) := \{ w \in Y \mid wR^Y w \} \]  
(diagonal)
Examples of modes:

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

\[ !X := \{ \langle w, x \rangle \mid w \in \mathcal{W}, x \in X \} \quad \text{(realizing)} \]

Examples of projections:

\[ \langle R \rangle := \{ w \in \mathcal{W} \mid \exists v \in \mathcal{W}. wR^\mathcal{W} v \} \quad \text{(domain)} \]

\[ \pi^{-1}(R) := \{ w \in \mathcal{W} \mid \exists v \in \mathcal{W}. vR^\mathcal{W} w \} \quad \text{(codomain)} \]

\[ \sim R := \{ w \in \mathcal{W} \mid \forall v \in \mathcal{W}. \neg (wR^\mathcal{W} v) \} \quad \text{(antidomain)} \]

\[ \Delta(R) := \{ w \in \mathcal{W} \mid wR^\mathcal{W} w \} \quad \text{(diagonal)} \]

**NOTE THAT:**

\[ \langle R \rangle = \sim \sim R \]
\[ = R/R\sim \cap \cdot \]

\[ \Delta(R) = R \cap \cdot \]

\[ \pi^{-1}(R) = \langle R\sim \rangle \]
Comments for logicians

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

The right algebraic two-sorted setting would be boolean modules over idempotent semirings. It is possible to move the discussion to one-sorted setting, though: Dynamic Relation Algebras (DRA's) studied in the 1990's by a Dutch group in Utrecht (A. Visser, M. Hollenberg) that is, idempotent semirings with antidomain operation $\sim$. The Utrecht group used the name dynamic negation 'Antidomain': term introduced recently by Desharnais, Jipsen and Struth.
Comments for logicians

• Note we do **not** allow transitive closure of arbitrary path expressions (allowed in non-standard extensions like Regular XPath)

• Note also that path expressions, as opposed to node expressions, are **not closed under other boolean connectives** than sum (changed in XPath 2.0)

• Therefore, we are **not exactly in the world of Tarski’s relation algebras**
Comments for logicians

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

• Note we do not allow transitive closure of arbitrary path expressions (allowed in non-standard extensions like Regular XPath)
• Note also that path expressions, as opposed to node expressions, are not closed under other boolean connectives than sum (changed in XPath 2.0)
• Therefore, we are not exactly in the world of Tarski’s relation algebras
• The right algebraic two-sorted setting would be boolean modules over idempotent semirings
• It is possible to move the discussion to one-sorted setting, though:
  Dynamic Relation Algebras (DRA’s) studied in the 1990’s by a Dutch group in Utrecht (A. Visser, M. Hollenberg)
Comments for logicians

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

   Dynamic Relation Algebras (DRA’s)
   studied in the 1990’s by a Dutch group in Utrecht (A. Visser, M. Hollenberg)
   that is, idempotent semirings with antidomain operation ~
   The Utrecht group used the name dynamic negation
   Antidomain: term introduced recently by Desharnais, Jipsen and Struth
Enter the Short CoreXPath (SCX) of de Rijke and Marx: one-sorted notational variant
Enter the Short CoreXPath (SCX) of de Rijke and Marx:
one-sorted notational variant

Syntax of Short Core XPath:

\[
\begin{align*}
  s & ::= \downarrow, \uparrow, \leftarrow, \Rightarrow \\
  a & ::= s \mid s^+ \\
  \text{exp} & ::= \cdot \mid a \mid \text{exp}/\text{exp} \mid \text{exp} \cup \text{exp}
\end{align*}
\]
Enter the **Short CoreXPath (SCX)** of de Rijke and Marx: one-sorted notational variant

Syntax of Short Core XPath:

```
\[ s ::= \downarrow, \uparrow, \leftarrow, \Rightarrow \\
a ::= s | s^+ \\
\text{exp} ::= \cdot | a | \text{exp/\exp} | \text{exp} \cup \text{exp} | ?p | \sim \text{exp} \quad (p \in \Sigma) \]
```
Enter the Short CoreXPath (SCX) of de Rijke and Marx:
one-sorted notational variant

Syntax of Short Core XPath:

\[ \text{s} ::= \downarrow, \uparrow, \leftarrow, \rightarrow \]
\[ \text{a} ::= \text{s} | \text{s}^+ \]
\[ \text{exp} ::= \cdot | \text{a} | \text{exp}/\text{exp} | \text{exp} \cup \text{exp} | \text{?p} | \sim \text{exp} \quad (p \in \Sigma) \]

Definition of a single/restricted axis fragment remains the same
Semantics of Short Core XPath

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

\[
[s]^T = R_s
\]

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

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

\[
[A/B]^T = \text{composition of } [A]^T \text{ and } [B]^T
\]

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

\[
[A[φ]]^T = \{(n, m) \in [A]^T \mid m \in [φ]^T\}
\]

nexpr: subsets of $N$

\[
[p]^T = \{n \in N \mid n \in V(p)\}
\]

\[
[φ \land ψ]^T = [φ]^T \cap [ψ]^T
\]

\[
[¬φ]^T = N \setminus [φ]^T
\]

\[
[⟨A⟩]^T = \text{domain of } [A]^T = \{n \mid (n, m) \in [A]^T\}
\]
Semantics of Short Core XPath

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

$[s]^T = R_s$

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

$[\cdot]^T = \text{the identity relation on } N$

$[A/B]^T = \text{composition of } [A]^T \text{ and } [B]^T$

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

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

$[\sim A]^T = \{(n, n) \in N^2 \mid \forall m.(n, m) \not\in [A]^T\}$
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:

\[
\begin{align*}
t(p) &= ?p \\
t(\langle A \rangle) &= \sim\sim t(A) \\
t(\phi \land \psi) &= \sim(\sim t(\phi) \cup \sim t(\psi)) \\
t(A[\phi]) &= t(A)/t(\phi)
\end{align*}
\]

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:

\[
\begin{align*}
t(p) &= \sim p \\
t(\langle A \rangle) &= \sim \sim t(A) \\
t(\phi \land \psi) &= \sim (\sim t(\phi) \cup \sim t(\psi)) \\
t(A[\phi]) &= t(A) / t(\phi)
\end{align*}
\]
other connectives being straightforward. Clearly

\[
\begin{align*}
[A]^T &= [t(A)]^T & \text{for all } A \in \text{pexpr} \\
[\cdot [\phi]]^T &= [t(\phi)]^T & \text{for all } \phi \in \text{nexpr}
\end{align*}
\]
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 $[P]^T = [Q]^T$
Problems Equivalent to Equivalence

- (because of presence of \( \lor \) and \( \bigcup \)):
  - containment for node expressions
  - containment for path expressions
Problems Equivalent to Equivalence

• (because of presence of \( \lor \) and \( \cup \)):
  • containment for node expressions
  • containment for path expressions
• (because of presence of \( \land \) and \( \neg \)):
  • satisfiability for node expressions
Problems Equivalent to Equivalence

- (because of presence of $\lor$ and $\lor$):
  - containment for node expressions
  - containment for path expressions
- (because of presence of $\land$ and $\neg$):
  - satisfiability for node expressions
- satisfiability for path expressions ...
Problems Equivalent to Equivalence

- (because of presence of $\lor$ and $\cup$):
  - containment for node expressions
  - containment for path expressions
- (because of presence of $\land$ and $\neg$):
  - satisfiability for node expressions
- satisfiability for path expressions . . .
  . . . this is a bit nontrivial
Problems Equivalent to Equivalence

• (because of presence of $\lor$ and $\cup$):
  • containment for node expressions
  • containment for path expressions

• (because of presence of $\land$ and $\neg$):
  • satisfiability for node expressions

• satisfiability for path expressions . . .
  . . . this is a bit nontrivial
  • reduction of satisfaction to equivalence:
    straightforward
Problems Equivalent to Equivalence

• (because of presence of $\lor$ and $\bigcup$):
  • containment for node expressions
  • containment for path expressions

• (because of presence of $\land$ and $\neg$):
  • satisfiability for node expressions

• satisfiability for path expressions . . .
  . . . this is a bit nontrivial
  • reduction of satisfaction to equivalence:
    straightforward
  • reduction of equivalence to satisfaction:
    requires *The Nasty Trick*
Which expressions are equivalent?

Let’s give it a try:
Which expressions are equivalent?

Let’s give it a try:

- is it true that
  \[ \equiv \uparrow/\downarrow? \]
Which expressions are equivalent?

Let’s give it a try:

- is it true that

  \( \equiv \uparrow/\downarrow? \)
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 $\uparrow/\downarrow \equiv \Leftarrow^+ \U \cdot \U \Rightarrow^+$?
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
  \[ \cdot \left<\downarrow\right> \equiv \downarrow/\uparrow \]
  and
  \[ \uparrow/\downarrow \equiv \leftarrow^+ \cup \cdot \left<\uparrow\right> \cup \Rightarrow^+? \]
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
  \[ \uparrow/\downarrow \equiv \leftrightarrow^+ \mathcal{U} \cdot \mathcal{U} \Rightarrow^+? \]
- One last try: how about
  \[ \sim\sim\downarrow \equiv \downarrow/\uparrow \]
  and
  \[ \uparrow/\downarrow \equiv \leftrightarrow^+ \mathcal{U} \sim\sim\uparrow \mathcal{U} \Rightarrow^+? \]
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
  \[ \uparrow/\downarrow \equiv \leftarrow^+ U \cdot U \Rightarrow^+? \]

• One last try: how about
  \[ \sim\sim\downarrow \equiv \downarrow/\uparrow \]
  and
  \[ \uparrow/\downarrow \equiv \leftarrow^+ U \sim\sim\uparrow U \Rightarrow^+? \]
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
Logicians and algebraists have long studied similar problems in a different disguise:

| logic:   | algebras: | databases: |
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>
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>
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:
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
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:
  - $P \equiv P$
  - $P \equiv Q = Q \equiv P$
  - $P \equiv Q = R \equiv P$ if $R$ is obtained from $R'$ by replacing occurrences of $P$ by $Q$.

Clearly, these rules preserve validity.
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:
  
  $P \equiv P$
  
  $P \equiv Q \quad \implies \quad Q \equiv P$
  
  $P \equiv Q \quad \land \quad Q \equiv R \quad \implies \quad P \equiv R$
  
  $(R' \equiv R \text{ is obtained from } R \text{ by replacing occurrences of } P \text{ by } Q)$

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:
  - $P \equiv P$
  - $P \equiv Q \implies Q \equiv P$
  - $P \equiv Q \& Q \equiv R \implies P \equiv R$

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:
  - $P \equiv P$
  - $P \equiv Q \implies Q \equiv P$
  - $P \equiv Q \land Q \equiv R \implies P \equiv R$
  - $P \equiv Q \implies R \equiv R'$

  ($R'$ is obtained from $R$ by replacing occurrences of $P$ by $Q$)
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:
  - $P \equiv P$
  - $P \equiv Q \implies Q \equiv P$
  - $P \equiv Q \land Q \equiv R \implies P \equiv R$
  - $P \equiv Q \implies R \equiv R'$
    (where $R'$ is obtained from $R$ by replacing occurrences of $P$ by $Q$)

- An axiomatization using Birkhoff’s rules only is **orthodox**.
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:
  
  \[
  \begin{align*}
  P & \equiv P \\
  P \equiv Q & \implies Q \equiv P \\
  P \equiv Q & \& Q \equiv R \implies P \equiv R \\
  P \equiv Q & \implies R \equiv R'
  \end{align*}
  \]

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

- 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
Before we proceed, you may have two questions:

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

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

- . . . the avalanche of results it triggered off?
Q1: Why Birkhoff Calculus?

Before we proceed, you may have two questions:

**Definition**

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

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

- . . . the avalanche of results it triggered off? 😊:
  *Theory of varieties* developed since the 1930’s: semigroups and groups, semirings, semilattices, lattices and residuated lattices, boolean algebras, abstract relation and cylindric algebras . . .
Q1 (ct’d): But What Use Are They For Us?
Q1 (ct’d): But What Use Are They For Us?

An orthodox axiomatization

≡

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

Almost all axiomatizations presented today will be orthodox
(you’re going to see one exception at the end of the talk and dislike it)
Q2: Anything Special about XPath?

Question

*How about complete axiomatizations for SQL-like languages?*

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

Question
How about complete axiomatizations for SQL-like languages?
After all, there has been nothing XML specific to what I said . . .

Answer
Even with no more than three attributes, you soon run into unaxiomatizability results! (©by logicians and algebraists)
Some database theorists got into problems not knowing about it . . .
Q2: Anything Special about XPath?

Question
How about complete axiomatizations for SQL-like languages?
After all, there has been nothing XML specific to what I said . . .

Answer
Even with no more than three attributes, you soon run into unaxiomatizability results! (© by logicians and algebraists)
Some database theorists got into problems not knowing about it . . .
It does not mean you cannot find interesting axiomatizable fragments—they are rather small though
Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer
Yes.

Long Answer
Yes, precisely because
• we can isolate the navigational core . . . (would not make much sense in the relational context)
• . . . and this core is related to well-behaved, axiomatizable formalisms:
  • Node expressions—to modal logic
  • Path expressions—to idempotent (antidomain) semirings
• The duality of path and node expressions:
  resembles (fragments of) the logic of programs (PDL)
Q2 (ct'd): Is XPath Querying Any Better Off, Then?

Short Answer
Yes.
Q2 (ct’d): Is XPath Querying Any Better Off, Then?

Short Answer
Yes.

Long Answer
Yes, precisely because

- *we can isolate the navigational core* . . .
  (would not make much sense in the relational context)
Q2 (ct’d): Is XPath Querying Any Better Off, Then?

Short Answer
Yes.

Long Answer
Yes, precisely because

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

• . . . and this core is related to
  well-behaved, axiomatizable formalisms:
    • Node expressions—to modal logic
Q2 (ct’d): Is XPath Querying Any Better Off, Then?

Short Answer
Yes.

Long Answer
Yes, precisely because

- we can isolate the navigational core . . .
  (would not make much sense in the relational context)
- . . . and this core is related to well-behaved, axiomatizable formalisms:
  - Node expressions—to modal logic
  - Path expressions—to idempotent (antidomain) semirings
Q2 (ct’d): Is XPath Querying Any Better Off, Then?

Short Answer
Yes.

Long Answer
Yes, precisely because

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

- . . . and this core is related to
  well-behaved, axiomatizable formalisms:
    - Node expressions—to modal logic
    - Path expressions—to idempotent (antidomain) semirings
    - The duality of path and node expressions:
      resembles (fragments of) the logic of programs (PDL)
Q2 (ct’d): Is XPath Querying Any Better Off, Then?

Short Answer
Yes.

Long Answer
Yes, precisely because

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

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\left\{\begin{array}{l}
\cdot/A \equiv A \\
A/\cdot \equiv A
\end{array}\right.
ISAx6\left\{\begin{array}{l}
A/(B \cup C) \equiv A/B \cup A/C \\
(A \cup B)/C \equiv A/C \cup B/C
\end{array}\right.
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 \( A[\phi \lor \psi] \equiv A[\phi] \cup A[\psi] \)

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

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  $A \left[ \neg \langle B \rangle \right] / B \equiv \bot$
PrAx2  $A[\phi \lor \psi] \equiv A[\phi] \cup A[\psi]$
PrAx3  $(A/B)[\phi] \equiv A/B[\phi]$
PrAx4  $\cdot[\langle \cdot \rangle] \equiv \cdot$

NdAx1  $\phi \equiv \neg(\neg \phi \lor \psi) \lor \neg(\neg \phi \lor \neg \psi)$
NdAx2  $\langle A \cup B \rangle \equiv \langle A \rangle \lor \langle B \rangle$
NdAx3  $\langle A/B \rangle \equiv \langle A[\langle B \rangle] \rangle$
NdAx4  $\langle \cdot [\phi] \rangle \equiv \phi$
Axioms in one-sorted signature

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

\[
\begin{align*}
\sim A/A & \equiv \bot \\
\sim \sim A/A & \equiv A \\
\sim (A/B)/A & \equiv (\sim (A/B)/A)/\sim B \\
\sim (A \cup B) & \equiv \sim A/\sim B \\
\sim A \cup \sim B & \equiv \sim \sim (\sim A \cup \sim B)
\end{align*}
\]

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(⇓), the child-axis-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-axis-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 Linear Axes

The non-transitive case:

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

This forces \textit{functionality} of the corresponding axis
Axioms for Transitive Axes

One for node expressions, one for path expressions:

TransAx1  \( \langle s^+ [\phi] \rangle \equiv \langle s^+ [\phi \land \neg \langle s^+ [\phi] \rangle] \rangle \)

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

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

for \( s \in \{\Rightarrow, \Leftarrow, \Uparrow\} \)

together with transitivity axioms

This forces the corresponding axis is a linear order
Single Axis Completeness Result

Theorem

- **Base axioms** are complete
  for Core XPath(\(\downarrow\))

- **Base axioms with LinAx1** are complete
  for other intransitive single axis fragments

- **Base axioms with TransAx1 and TransAx2** are complete
  for Core XPath(\(\downarrow^+\))

- **Base axioms with TransAx1, TransAx2 and LinAx2** are complete
  for other transitive single axis fragments
A Few Words About Proofs

- First, rewrite node expressions to simple node expressions:

\[ \text{siNode} ::= \langle \cdot \rangle \mid p \mid \langle a \text{[siNode]} \rangle \mid \neg \text{siNode} \mid \text{siNode} \lor \text{siNode} \]
A Few Words About Proofs

- First, rewrite node expressions to simple node expressions:

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

They are isomorphic variants of modal formulas.
A Few Words About Proofs

• First, rewrite node expressions to simple node expressions:

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

They are isomorphic variants of modal formulas

• Using normal form theorems for modal logic, we provide a completeness proof for node expressions
Then we rewrite all path expressions as sums of sum-free expressions of the form

\[ S = \cdot [\beta_1] / a [\beta_2] / \ldots / a [\beta_\ell], \]

(all \( \beta_i \) 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.
Then we rewrite all path expressions as sums of sum-free expressions of the form

\[ S = \cdot [\beta_1] / a [\beta_2] / \ldots / a [\beta_\ell], \]

(all \( \beta_i \) 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
Then we rewrite all path expressions as sums of sum-free expressions of the form

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

(all $\beta_i$ 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
Then we rewrite all path expressions as sums of sum-free expressions of the form

\[ S = \cdot [\beta_1] / a [\beta_2] / \ldots / a [\beta_\ell], \]

(all \( \beta_i \) 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

\[ p \land q \equiv \bot \]

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

\[ p \land q \equiv \bot \]

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
Axiom For Axes Dependencies

TreeAx1
\[
\begin{align*}
    s^+/s \cup s & \equiv s^+ \\
    s/s^+ \cup s & \equiv s^+
\end{align*}
\]

TreeAx2
\[
\begin{align*}
    s[\phi]/s\sim & \equiv \cdot [\langle s[\phi] \rangle] \text{ (for s distinct than } \uparrow) \\
\end{align*}
\]

TreeAx3
\[
\begin{align*}
    \uparrow[\phi]/\downarrow & \equiv (\leftarrow^+ \cup \Rightarrow^+ \cup \cdot) [\langle \uparrow[\phi] \rangle] \\
\end{align*}
\]

TreeAx4
\[
\begin{align*}
    \leftarrow^+ & \equiv \leftarrow^+ [\langle \uparrow \rangle] \\
    \Rightarrow^+ & \equiv \Rightarrow^+ [\langle \uparrow \rangle]
\end{align*}
\]

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)

\[
\begin{align*}
\langle s [\neg \langle \cdot \rangle] \rangle & \equiv \neg \langle \cdot \rangle \\
\langle s [\phi \lor \psi] \rangle & \equiv \langle s [\phi] \rangle \lor \langle s [\psi] \rangle \\
\phi & \leq \neg \langle s [\neg \langle s \neg [\phi] \rangle] \rangle \\
\langle s [\neg \phi] \rangle \land \langle s [\phi] \rangle & \equiv \neg \langle \cdot \rangle \quad \text{(for } s \text{ distinct than } \uparrow) \\
\langle s [\phi] \rangle \lor \langle s [\langle s^+ [\phi] \rangle] \rangle & \equiv \langle s^+ [\phi] \rangle \\
\neg \langle s [\phi] \rangle \land \langle s^+ [\phi] \rangle & \leq \langle s^+ [\neg \phi \land \langle s [\phi] \rangle] \rangle \\
\langle s [\langle \cdot \rangle] \rangle & \leq \langle s^+ [\neg \langle s [\langle \cdot \rangle] \rangle] \rangle \\
\text{TransAx1 for } \downarrow^+ \text{ and } \Rightarrow^+ \\
\langle \downarrow [\neg \langle \leftrightarrow \rangle \land \neg \langle \Rightarrow^* [\phi] \rangle] \rangle & \leq \neg \langle \downarrow [\phi] \rangle \\
\langle \downarrow [\phi] \rangle & \leq \langle \downarrow [\neg \langle \leftrightarrow \rangle] \rangle \land \langle \downarrow [\neg \langle \Rightarrow \rangle] \rangle \\
\neg \langle \uparrow \rangle & \leq \neg \langle \leftrightarrow \rangle \land \neg \langle \Rightarrow \rangle
\end{align*}
\]
The Nasty Trick

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

Add the separability rule:

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

Except for spoiling the whole equational story,
We can use this to provide an axiomatization for path expressions . . .
We can use this to provide an axiomatization for path expressions . . .

. . . of a sort—a non-orthodox one! 🙂
The Nasty Trick

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

. . . of a sort—a non-orthodox one! 😞

Add the separability rule:

\[(\text{Sep}) \text{ IF } \langle A [\rho] \rangle \equiv \langle B [\rho] \rangle \text{ for } \rho \text{ not occurring in } A, B \]

\[\text{THEN } A \equiv B.\]
We can use this to provide an axiomatization for path expressions . . .

. . . of a sort—a non-orthodox one! Add the separability rule:

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

THEN \( A \equiv B \).

Except for spoiling the whole equational story, it does not sit too well with the labelling axiom . . .
The Nasty Trick Does Its Job

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

- Query equivalence of Core XPath(⇒+, ⇑+, ⇒), Core XPath(⇑+), Core XPath(s) (for s ∈ {⇑, ⇐, ⇒}) is coNP-complete.
- Query equivalence of Core XPath(⇐+, ⇐, ⇒+, ⇒, ↑+, ↑) is PSPACE-complete.
  Thus, the PSPACE upper bound applies to all its sublanguages.
- Query equivalence of Core XPath(⇓) and Core XPath(⇓+) is PSPACE-complete.
  Thus, all extensions of this fragment are PSPACE-hard.
- Query equivalence of Core XPath(⇓, ⇓+) is EXPTIME-complete.
  Thus, all extensions of this fragment are EXPTIME-hard.
Proofs

...by reductions to complexity results for modal logics like K, GL, Alt.1 and fragments of tense/temporal logic on linear and branching orders. The most interesting one is for the second clause—somewhat tricky embedding into a logic of Sistla and Clarke.
What we have seen so far . . .

We have seen:

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

- Computational complexity results for path equivalences in most meaningful sublanguages of Core XPath
What we have seen so far . . .

We have seen:

- equational axiomatizations for path equivalences of all eight single axis fragments of Core XPath
- equational axiomatizations for node equivalences of full Core XPath 1.0
What we have seen so far . . .

We have seen:

- **equational axiomatizations for path equivalences** of all eight **single axis fragments** of Core XPath
- **equational axiomatizations** for **node equivalences** of full Core XPath 1.0
- **non-orthodox axiomatization** for **path equivalences** of full Core XPath 1.0
What we have seen so far . . .

We have seen:

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

• **equational axiomatizations for node equivalences of full Core XPath 1.0**

• **non-orthodox axiomatization for path equivalences of full Core XPath 1.0**

• **computational complexity results for path equivalences in most meaningful sublanguages of Core XPath 1.0**
What we have not seen so far . . .

- Definability and expressivity results
- Results for stronger fragments of XPath
Possible yardsticks for expressive power on trees:
- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... e.g., in between FO and MSO lies FO(TC)
Expressive power

Possible yardsticks for expressive power on trees:

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

What kind of queries do we want to characterize?

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

Possible types of characterizations:

- Syntactic (e.g. "\( L \) is equivalent to the two variable ...")
- Semantic (e.g., "bisimulation invariant fragment ...")

Decidable characterizations?
Possible yardsticks for expressive power on trees:

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

What kind of queries do we want to characterize?

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

Possible yardsticks for expressive power on trees:

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

What kind of queries do we want to characterize:

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

Possible types of characterizations:
Expressive power

Possible yardsticks for expressive power on trees:
- First-order logic (FO) (cf. Codd completeness of SQL/RA)
- Monadic second-order logic (MSO)
- ... — e.g., in between FO and MSO lies FO(TC)

What kind of queries do we want to characterize
- Binary relations definable by path expressions?
- Node sets definable by node expressions?
- Properties of trees definable by node expressions evaluated at the root?

Possible types of characterizations:
- **Syntactic** (e.g. “L is equivalent to the two variable ...”) versus **semantic** (e.g., “bisimulation invariant fragment ...”)
Expressive power

Possible yardsticks for expressive power on trees:

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

What kind of queries do we want to characterize

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

Possible types of characterizations:

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

Decidable characterizations?
Descendant-only fragment

_CoreXPath_\(\downarrow^+\) node expressions have the same expressive power as MSO formulas \(\varphi(x)\) for which

(i) truth of \(\varphi(x)\) at a node depends only on the subtree

(ii) \(\varphi(x)\) does not distinguish children from descendants, i.e., the following operation preserves truth/falsity at the root:

\[ \text{Easy proof from De Jongh-Sambin fixed point theorem for GL} \]
\[ \text{and Janin-Walukiewicz theorem for } \mu \text{-calculus} \]
\[ \text{Moreover, the proof is effective: it yields a decision procedure.} \]
\[ \text{A variant for path expressions obtained by replacing } \mu \text{-formulas by } \mu \text{-programs} \]
\[ \text{bisimulation invariance by bisimulation safety} \]
Expressive power (ct’d)

Descendant-only fragment

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

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

Easy proof from De Jongh-Sambin fixed point theorem for GL and Janin-Walukiewicz theorem for $\mu$-calculus
Descendant-only fragment

CoreXPath(↓⁺) node expressions have the same expressive power as MSO formulas \( \varphi(x) \) for which

(i) truth of \( \varphi(x) \) at a node depends only on the subtree

(ii) \( \varphi(x) \) does not distinguish children from descendants, i.e., the following operation preserves truth/falsity at the root:

Easy proof from De Jongh-Sambin fixed point theorem for GL and Janin-Walukiewicz theorem for \( \mu \)-calculus

Moreover, the proof is effective: it yields a decision procedure.
Descendant-only fragment

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

(i) truth of \( \varphi(x) \) at a node depends only on the subtree

(ii) \( \varphi(x) \) does not distinguish children from descendants, i.e.,

the following operation preserves truth/falsity at the root:

Easy proof from De Jongh-Sambin fixed point theorem for GL
and Janin-Walukiewicz theorem for \( \mu \)-calculus

Moreover, the proof is effective: it yields a decision procedure.
A variant for path expressions obtained by replacing

\( \mu \)-formulas by \( \mu \)-programs
bisimulation invariance by bisimulation safety
Expressive power (ct’d)

What about the full Core XPath language?
Expressive power (ct’d)

What about the full Core XPath language?

No decidable characterization in terms of MSO is known. (but see a LICS 2010 result of Place/Segoufin for a decidable characterization of CoreXPath($\downarrow^+, \uparrow^+, \leftarrow^+, \Rightarrow^+$) in terms of forest algebras)
What about the full Core XPath language?

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

Syntactic characterization of full CXP (Marx-De Rijke 05)
Core XPath node expressions have the same expressive power as formulas $\phi(x)$ in the two-variable fragment of $FO[R\ll, R\ll^+, R\Rightarrow, R\Rightarrow+]$. There is a similar characterization for path expressions.
Brief recap

In summary...  

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


In summary... 

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

- In the remainder of this talk, we consider two extensions:
In summary...

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

- In the remainder of this talk, we consider two extensions:
  - Regular XPath: the extension of Core XPath with full transitive closure.
In summary…

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

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

Extending XPath with transitive closure proposed for various reasons.


• Core XPath extended with transitive closure has full first-order expressive power

• is rich enough to express DTDs and

• admits view-based query rewriting with recursive views

• very natural from the perspective of PDL

Part of community standard EXSLT

Implemented (naively) in Saxon and Xalan

. . . but left out by W3C from XPath 2.0
Motivation for Regular XPath

Extending XPath with transitive closure proposed for various reasons.

- Practical reasons: some applications require the use of transitive closure.
**Motivation for Regular XPath**

Extending XPath with transitive closure proposed for various reasons.

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

- **Core XPath extended with transitive closure**
  - has full first-order expressive power
  - is rich enough to express DTDs and
  - admits view-based query rewriting with recursive views
Extending XPath with transitive closure proposed for various reasons.

- Practical reasons: some applications require the use of transitive closure.
- Core XPath extended with transitive closure
  - has full first-order expressive power
  - is rich enough to express DTDs and
  - admits view-based query rewriting with recursive views
- very natural from the perspective of PDL
Motivation for Regular XPath

Extending XPath with transitive closure proposed for various reasons.

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

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

- Regular XPath has two types of expressions:

  • path expressions
    \[ \alpha ::= \uparrow | \downarrow | \leftarrow | \rightarrow | \alpha/\beta | \alpha \cup \beta | \alpha^* | \alpha[\phi] \]

  • node expressions
    \[ \phi ::= p | \neg \phi | \phi \land \psi | \langle \alpha \rangle \]
Syntax of Regular XPath

- Regular XPath has two types of expressions:
  - path expressions
    \[
    \alpha ::= \uparrow \mid \downarrow \mid \leftarrow \mid \rightarrow \mid . \mid \alpha/\beta \mid \alpha \cup \beta \mid \alpha^* \mid \alpha[\phi]
    \]
Syntax of RegularXPath

- Regular XPath has two types of expressions:
  - path expressions
    $$\alpha ::= \uparrow | \downarrow | \Leftarrow | \Rightarrow | . | \alpha/\beta | \alpha \cup \beta | \alpha^* | \alpha[\phi]$$
  - node expressions
    $$\phi ::= p | \neg \phi | \phi \land \psi | \langle \alpha \rangle$$
“Go to the next book that has at least two authors.”

In Regular XPath:
“Go to the next book that has at least two authors.”

In Regular XPath:

\[
(⇒[¬\text{twoauthorbook}])* / ⇒[\text{twoauthorbook}]
\]

where \text{twoauthorbook} stands for
\text{book} \land (\downarrow[\text{author}] / ⇒^+[\text{author}]).
Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that
Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

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

Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

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

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

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

- Let root be short for \(\neg\langle \uparrow \rangle\).
  Let leaf be short for \(\neg\langle \downarrow \rangle\).
  Let first be short for \(\neg\langle \leftarrow \rangle\).
  Let last be short for \(\neg\langle \Rightarrow \rangle\).

- Let suc be shorthand for \(\downarrow[\text{first}] \cup .[\text{leaf}]/(\uparrow \text{ while } \text{last})/\Rightarrow\) (the successor in depth first left-to-right ordering).
Another example

The following can be expressed in Regular XPath:

“The tree has an even number of nodes”

To see this, note that

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

Consider game trees:

- leafs are labeled by Anne-wins or Bob-wins
- inner nodes are labeled by Anne’s-move or Bob’s-move
One more example

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

- Puzzle:
  Show that “Anne has a winning strategy” is expressible.
Expressive power of Regular XPath

• What is the expressive power of Regular XPath?

• We know that $\mathcal{FO} \subseteq \text{Regular XPath} \subseteq \mathcal{FO}(\text{TC})$ (The first inclusion follows from results by Marx 2004).

• A natural conjecture: $\text{Regular XPath} \equiv \mathcal{FO}(\text{TC})$ (after all, Regular XPath has a transitive closure operator!)

• Ten Cate and Segoufin managed to prove this only after extending Regular XPath with a "within" operator $W$:

$$T_n, n \mid = W \phi \iff T_n, n \mid = \phi$$

(cf. temporal logics with forgettable past)
Expressive power of Regular XPath

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

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

(The first inclusion follows from results by Marx 2004).
Expressive power of Regular XPath

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

\[ FO \subsetneq Regular \ XPath \subseteq FO(TC) \]

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

- A natural conjecture:

\[ Regular \ XPath \equiv FO(TC) \]

(after all, Regular XPath has a transitive closure operator!)
Expressive power of Regular XPath

• What is the expressive power of Regular XPath?

• We know that

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

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

• A natural conjecture:

\[ \text{Regular XPath} \equiv FO(TC) \]

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

• Ten Cate and Segoufin managed to prove this only after extending Regular XPath with a “within” operator \( W \):

\[ T, n \models W\phi \iff T_n, n \models \phi \]

(cf. temporal logics with forgettable past)
Expressive power of Regular XPath (ct’d)

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

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

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

Similarly for node expressions.

- **Corollary:** Regular XPath(W) is closed under path intersection and complementation.
Expressive power of Regular XPath (ct’d)

- $FO(TC)$ is the extension of first-order logic with a transitive closure operator for binary relations.
Expressive power of Regular XPath (ct’d)

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

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

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

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

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

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

• **Corollary:** Regular XPath($W$) is closed under path intersection and complementation.
Axiomatizations and complexity

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

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

As for complexity,
Axiomatizations and complexity

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

As for complexity,

- Query evaluation can still be performed in \textit{PTime} even for Regular XPath(W).

- Query containment is still \textit{ExpTime-complete} for Regular XPath but it is \textit{2ExpTime-complete} for Regular XPath(W).
Core XPath 2.0
XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection and complementation** of path expressions.
  \[ \alpha \text{ intersect } \beta \quad \text{and} \quad \alpha \text{ except } \beta \]

Example:
\[
\ast \[p\] \text{ except } \ast \[q\]/\ast \[p\]
\]
XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection and complementation** of path expressions.
  \( \alpha \text{ intersect } \beta \) and \( \alpha \text{ except } \beta \)

**Example:** \( \downarrow^* [p] \text{ except } \downarrow^* [q] / \downarrow^* [p] \)
XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection and complementation** of path expressions.
  \[\alpha \text{ intersect } \beta \text{ and } \alpha \text{ except } \beta\]

  Example: \(\downarrow^*[p] \text{ except } \downarrow^*[q]/\downarrow^*[p]\)

- **Variables and for loops**

  for $x$ in \(\alpha\) return \(\beta\) and 
  \(\alpha[. \text{ is } x]\)
XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection and complementation** of path expressions.
  \[ \alpha \text{ intersect } \beta \quad \text{and} \quad \alpha \text{ except } \beta \]
  
  Example: \[ \downarrow^*[p] \text{ except } \downarrow^*[q]/\downarrow^*[p] \]

- **Variables and for loops**
  
  for $x$ in $\alpha$ return $\beta$ and $\alpha[. \text{ is } x]$
  
  Example:
  
  for $x$ in . return $\downarrow^*[p \land \neg(\uparrow^*[q]/\uparrow^*[. \text{ is } x])]$
XPath 2.0 extends XPath 1.0 with many features, including the following new navigational operations:

- **Intersection and complementation** of path expressions. 
  \[ \alpha \text{ intersect } \beta \text{ and } \alpha \text{ except } \beta \]
  
  **Example:** \( \downarrow^* [p] \text{ except } \downarrow^*[q]/\downarrow^*[p] \)

- **Variables and for loops**
  
  for \( x \) in \( \alpha \) return \( \beta \) and \( \alpha[. \text{ is } x] \)
  
  **Example:**
  for \( x \) in . return \( \downarrow^*[p \land \neg(\uparrow^*[q]/\uparrow^*[. \text{ is } x])] \)

- **Core XPath 2.0** is the extension of Core XPath with these features.
The path intersection and complementation turn Core XPath 2.0 into a version of Tarski’s relation algebra (interpreted on finite ordered trees).
• The path intersection and complementation turn Core XPath 2.0 into a version of Tarski’s relation algebra (interpreted on finite ordered trees).

• The **variables** and **for-loops** make it possible to give a linear translation from first-order logic to Core XPath 2.0:

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

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

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

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

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

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

\[
TR'(\phi \land \psi) = TR'(\phi) \land TR'(\psi)
\]

\[
TR'(\neg \phi) = \neg TR'(\phi)
\]

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

where \( \top \) is shorthand for \( \uparrow^*/\downarrow^* \) (the universal relation)
• Core XPath 2.0 has the same expressive power as first-order logic, both with and without variables (in the case with variables there is a linear translation).
Expressivity, Complexity and Axiomatization

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

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

• The complexity of the query equivalence problem is non-elementary, both with and without variables.

  (even adding only path intersection to Core XPath makes it 2ExpTime-complete.)
Expressivity, Complexity and Axiomatization

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

- The complexity of the **query equivalence** problem is **non-elementary**, both with and without variables.

  (even adding only **path intersection** to Core XPath makes it 2ExpTime-complete.)

- We have two **complete axiomatizations** of path equivalence in Core XPath 2.0: one with and one without variables.
The case without variables

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

- Recall that, without variables, Core XPath 2.0 is essentially a version of Relation Algebra interpreted on finite sibling ordered trees.
- One apparent problem: Relation Algebra has no node tests. However, these can easily be translated away:

\[
\begin{align*}
\text{Pred1. } & \alpha[\phi \land \psi] \equiv \alpha[\phi][\psi] \\
\text{Pred2. } & \alpha[\phi \lor \psi] \equiv \alpha[\phi] \cup \alpha[\psi] \\
\text{Pred3. } & \alpha[\neg \phi] \equiv \alpha - \alpha[\phi] \\
\text{Pred4. } & \alpha[\langle \beta \rangle] \equiv \alpha/((\beta/\top) \cap .)
\end{align*}
\]
The case without variables

- Recall that, without variables, Core XPath 2.0 is essentially a version of Relation Algebra interpreted on finite sibling ordered trees.
- One apparent problem: Relation Algebra has no node tests. However, these can easily be translated away:

\[
\begin{align*}
\text{Pred1.} \quad &\alpha[\phi \land \psi] \equiv \alpha[\phi][\psi] \\
\text{Pred2.} \quad &\alpha[\phi \lor \psi] \equiv \alpha[\phi] \cup \alpha[\psi] \\
\text{Pred3.} \quad &\alpha[\neg \phi] \equiv \alpha - \alpha[\phi] \\
\text{Pred4.} \quad &\alpha[\langle \beta \rangle] \equiv \alpha/((\beta/\top) \cap .)
\end{align*}
\]

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

- **BA1.** \( \alpha \cup (\beta \cup \gamma) \equiv (\alpha \cup \beta) \cup \gamma \)
- **BA2.** \( \alpha \cap (\beta \cap \gamma) \equiv (\alpha \cap \beta) \cap \gamma \)
- **BA3.** \( \alpha \cup \beta \equiv \beta \cup \alpha \)
- **BA4.** \( \alpha \cap \beta \equiv \beta \cap \alpha \)
- **BA5.** \( \alpha \cup (\beta \cap \gamma) \equiv (\alpha \cup \beta) \cap (\alpha \cup \gamma) \)
- **BA6.** \( \alpha \cap (\beta \cup \gamma) \equiv (\alpha \cap \beta) \cup (\alpha \cap \gamma) \)
- **BA7.** \( \alpha \cup (\alpha \cap \beta) \equiv \alpha \)
- **BA8.** \( \alpha \cap (\alpha \cup \beta) \equiv \alpha \)
- **BA9.** \( \alpha \cup (\top - \alpha) \equiv \top \)
- **BA10.** \( \alpha \cap (\top - \alpha) \equiv \bot \)
- **BA11.** \( \alpha \cap (\top - \beta) \equiv \alpha - \beta \)
The axioms of Relation algebra

- RA1. \( \alpha/(\beta/\gamma) \equiv (\alpha/\beta)/\gamma \)
- RA2. \( \alpha/\cdot \equiv \alpha \)
- RA3. \( (\alpha \cup \beta)/\gamma \equiv \alpha/\gamma \cup \beta/\gamma \)
- RA4. \( (\alpha \cup \beta)^\sim \equiv \alpha^\sim \cup \beta^\sim \)
- RA5. \( (\alpha/\beta)^\sim \equiv \beta^\sim /\alpha^\sim \)
- RA6. \( (\alpha^\sim )^\sim \equiv \alpha \)
- RA7. \( (\alpha/(\top - (\alpha^\sim /\beta))) \subseteq \top \text{ except } \beta \)

To completely axiomatize relation algebra, normally, one needs to add also Venema’s Rule:

> If \( X \) is a relation variable not occurring in \( \alpha \) and \( X - (((\top - .)/X/\top) \cup (\top/X/(\top - .))) \subseteq \alpha \) then \( \alpha \equiv \top \).

Fortunately, this rule turns out to be derivable in our case.
The axioms for finite sibling ordered trees

\( \text{Tr1.} \) \( \downarrow^+ / \downarrow^+ \) \( \subset \) \( \downarrow^+ \)
\( \text{Tr2.} \) \( \downarrow^+ \cap \uparrow^+ \) \( \equiv \bot \)
\( \text{Tr3a.} \) \( \downarrow^+ \) \( \equiv \downarrow^+ \cup (\downarrow^+ / \downarrow^+) \)
\( \text{Tr3b.} \) \( \downarrow \) \( \equiv \downarrow^+ - (\downarrow^+ / \downarrow^+) \)
\( \text{Tr4.} \) \( [\langle \uparrow \rangle] \) \( \equiv [\langle \uparrow \rangle] \cdot [\langle \downarrow \rangle] \cup [\langle \downarrow \rangle] / \uparrow^+ \)
\( \text{Tr5.} \) \( \downarrow^+ / \uparrow^+ \) \( \subset \) \( \Rightarrow^+ \)
\( \text{Tr6.} \) \( \Rightarrow^+ / \Rightarrow^+ \) \( \equiv \bot \)
\( \text{Tr7.} \) \( \Rightarrow^+ \cap \Leftarrow^+ \equiv \bot \)
\( \text{Tr8a.} \) \( \Rightarrow^+ \) \( \equiv \Rightarrow^+ \cup (\Rightarrow^+ / \Rightarrow^+) \)
\( \text{Tr8b.} \) \( \Rightarrow \) \( \equiv \Rightarrow^+ - (\Rightarrow^+ / \Rightarrow^+) \)
\( \text{Tr9.} \) \( [\Leftarrow] \) \( \equiv [\langle \Leftarrow \rangle] \cdot [\langle \downarrow \rangle] \cup [\langle \downarrow \rangle] / \Rightarrow^+ \)
\( \text{Tr10.} \) \( \Rightarrow^+ \cup \Leftarrow^+ \equiv (\Rightarrow / \downarrow) - . \)
\( \text{Tr11.} \) \( \cdot \cup \Rightarrow^+ \cup \downarrow^+ \cup \)
\( \left(\uparrow^*/\Rightarrow^+ / \downarrow^*\right) \cup \left(\uparrow^*/\Leftarrow^+ / \downarrow^*\right) \)
\( \equiv \top \)
\( \text{Ind.} \) \( \top[\langle \alpha \rangle] \)
\( \equiv \top[\langle \alpha - (\alpha/ \Leftarrow) \rangle] \)
Rounding up

Three languages:

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

• Regular XPath(W): the extension with ∗ and W.
  Expressivity: same as FO(TC)
  Query evaluation: PTime
  Query equivalence: 2ExpTime-complete

• Core XPath 2.0: the navigational core of XPath 2.0
  Expressivity: same as FO.
  Query evaluation: PSpace-complete
  Query equivalence: non-elementary hard
Rounding up

Three languages:

- **Core XPath**: the navigational core of XPath 1.0
  
  Expressivity: $\text{FO}^2$
  
  Query evaluation: PTime
  
  Query equivalence: ExpTime-complete
Rounding up

Three languages:

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

- **Regular XPath(\(W\))**: the extension with $\ast$ and \(W\).
  Expressivity: same as $\text{FO}(\text{TC})$
  Query evaluation: PTime
  Query equivalence: 2ExpTime-complete
Rounding up

Three languages:

- **Core XPath**: the navigational core of XPath 1.0
  - Expressivity: FO²
  - Query evaluation: PTime
  - Query equivalence: ExpTime-complete

- **Regular XPath(W)**: the extension with * and W.
  - Expressivity: same as FO(TC)
  - Query evaluation: PTime
  - Query equivalence: 2ExpTime-complete

- **Core XPath 2.0**: the navigational core of XPath 2.0
  - Expressivity: same as FO.
  - Query evaluation: PSpace-complete
  - Query equivalence: non-elementary hard
Some references

Expressive power:

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


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

Axiomatization:


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

Complexity:

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

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