B4M36DS2, BE4M36DS2: **Database Systems 2**

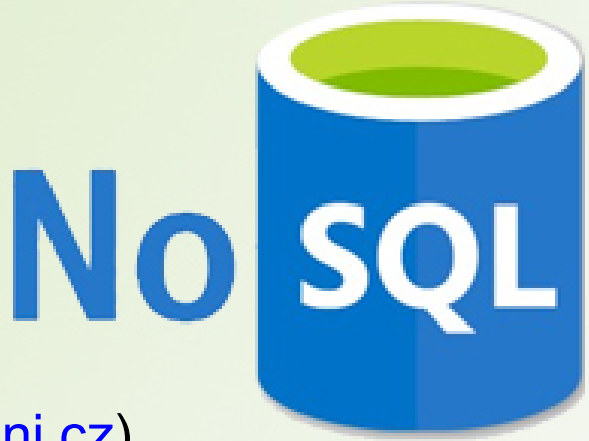
https://cw.fel.cvut.cz/b221/courses/b4m36ds2/

Lecture 3

XML Databases: XPath

Yuliia Prokop
prokoyul@fel.cvut.cz

3. 10. 2022



Author: Martin Svoboda (martin.svoboda@matfyz.cuni.cz)

Czech Technical University in Prague, Faculty of Electrical Engineering

Lecture Outline

XPath and XQuery

- Data model
- Query expressions
  - **Path expressions**
  - **Comparison expressions**
  - Variable assignments
  - Iteration expressions
  - Set operations
  - …

XPath

XML Path Language

Introduction

XPath = *XML Path Language*

- **Navigation and selection of nodes**
- Versions: 1.0 (1999), 2.0 (2010), 3.0 (2014), **3.1** (March 2017)
- W3C recommendation
  - https://www.w3.org/TR/xpath-31/

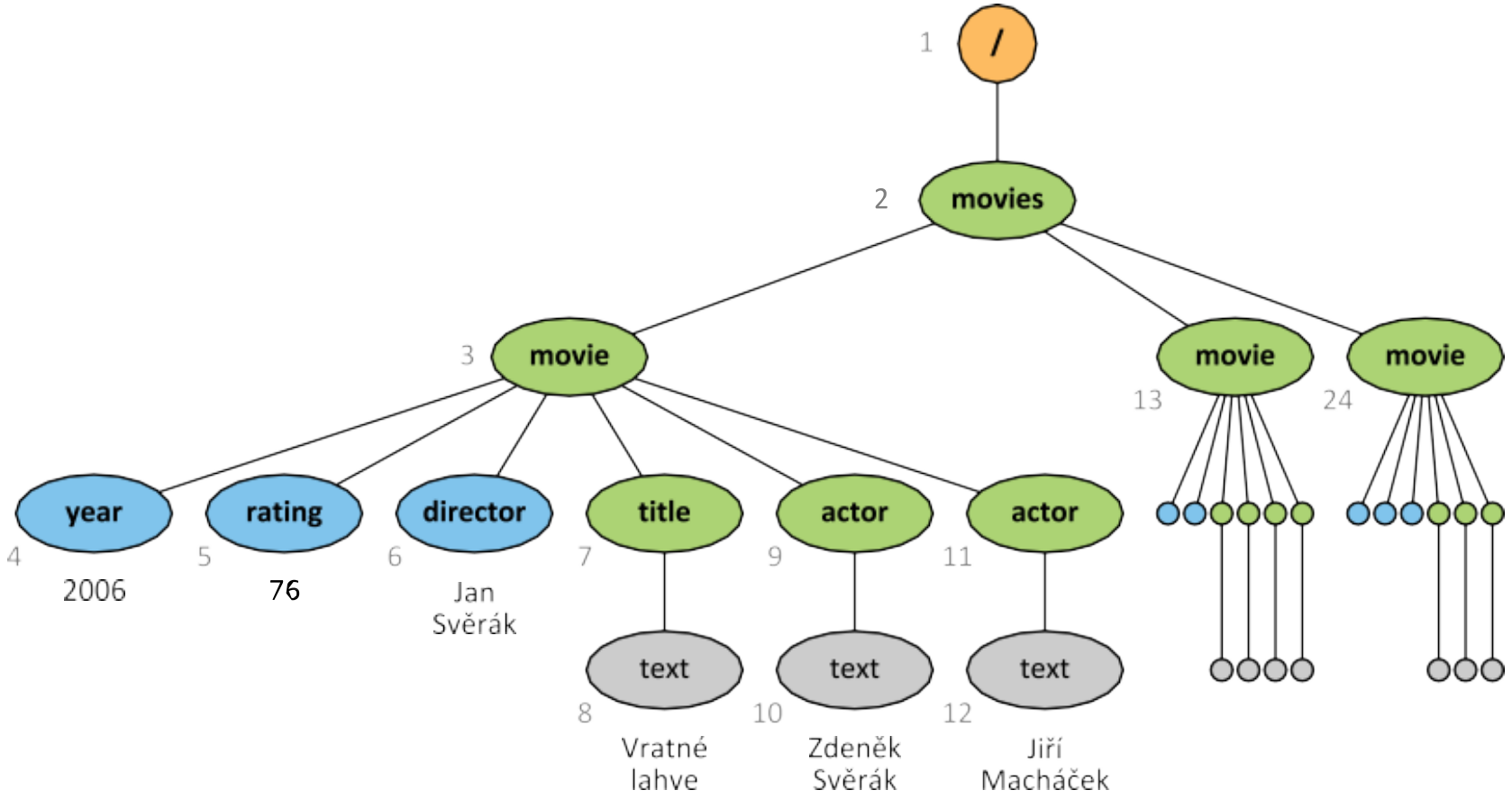
XQuery = *XML Query Language*

- **Complex queries and transformations**
- Contains XPath
- Versions: 1.0 (2007), 3.0 (2014), **3.1** (March 2017)
- W3C recommendation
  - https://www.w3.org/TR/xquery-31/

Sample Data

```xml
<?xml version="1.1" encoding="UTF-8"?>
<movies>
  <movie year="2006" rating="76" director="Jan Svěrák">
    <title>Vratné lahve</title>
    <actor>Zdeněk Svěrák</actor>
    <actor>Jiří Macháček</actor>
  </movie>
  <movie year="2000" rating="84">
    <title>Samotáři</title>
    <actor>Jitka Schneiderová</actor>
    <actor>Ivan Trojan</actor>
    <actor>Jiří Macháček</actor>
  </movie>
  <movie year="2007" rating="53" director="Jan Hřebejk">
    <title>Medvídek</title>
    <actor>Jiří Macháček</actor>
    <actor>Ivan Trojan</actor>
  </movie>
</movies>
```

Sample Data


1
/
2
movies
3
movie
13
movie
24
movie
4
year
2006
5
rating
76
6
director
Jan
Svěrák
7
title
9
actor
11
actor
8
text
Vratné
lahve
10
text
Zdeněk
Svěrák
12
text
Jiří
Macháček

Data Model

XDM = *XQuery and XPath Data Model* (XPath 2.0, XQuery 1.0)

- **XML tree** consisting of **nodes** of different kinds
  - Document, element, attribute, text, …
- **Document order**
  - The order in which nodes appear in the XML file
    - I.e. nodes are numbered using a **pre-order depth-first traversal**
- Reverse document order

Query result

- Each query expression is evaluated to a sequence

Data Model

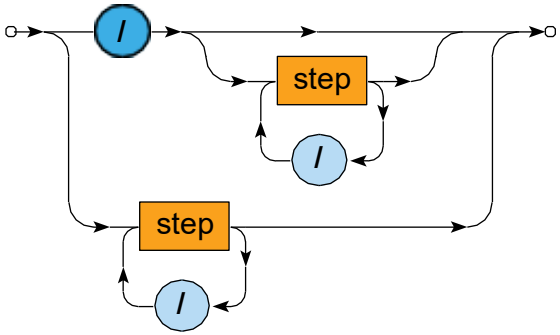
Sequence = **ordered collection of nodes and/or atomic values**

- Can be **mixed**
  - But usually just nodes, or just atomic values
- Are automatically **flattened**
  - E.g.: $(2,\ (),\ (4,\ 1,\ (3)),\ (1)) \Leftrightarrow (2,\ 4,\ 1,\ 3,\ 1)$
- Can be **empty**
  - E.g.: $()$
- Standalone items are treated as **singleton sequences**
  - E.g.: $1 \Leftrightarrow (1)$
- Can have **duplicate items**

Path Expressions

Path expression

- Allows for navigation within an XML tree
- Consists of **navigational steps**



- **Absolute** paths: start with /
  - Navigation starts at the **document node**
- **Relative** paths
  - Navigation starts at an implicitly specified context node

Path Expressions: Examples

Absolute path expressions

/
/movies
/movies/movie
/movies/movie/title/text()
/movies/movie/@year

Relative path expressions

actor/text()
@director

Path Expressions

Evaluation of a path expression P

- … with respect to the initial **context sequence** C

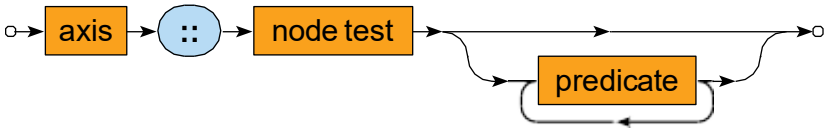
---

1 **if** P does not contain any step **then**
2 ⌊ **return** C (we already have the final result)
3 **else** (when P contains at least one step)
4 ⌊ let S be the first step and P' the remaining steps (if any)
5 ⌊ let $C' = \langle\rangle$ be an empty sequence
6 ⌊ **foreach** context node $u \in C$ **do**
7 ⌊ ⌊ evaluate S with respect to u and add the selected items C_u' to C'
8 ⌊ **return** evaluate P' with respect to C'

Path Expressions: Steps

Navigational step

- Each step consists of up to 3 components

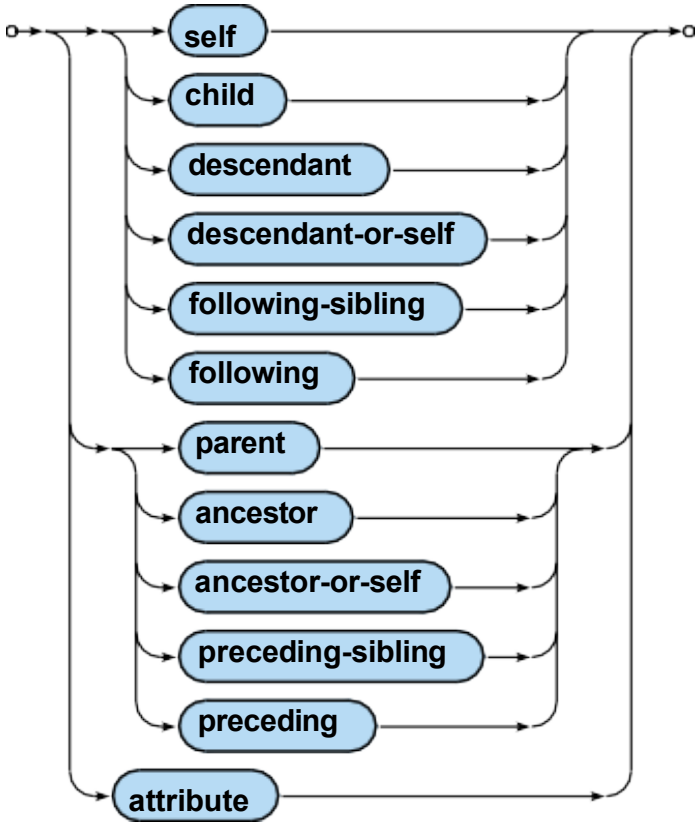


- **Axis**
  - **Relation of nodes** to be selected for a given context node u
- **Node test**
  - **Basic condition** these selected nodes must satisfy
- **Predicates**
  - **Advanced conditions** these nodes must further satisfy

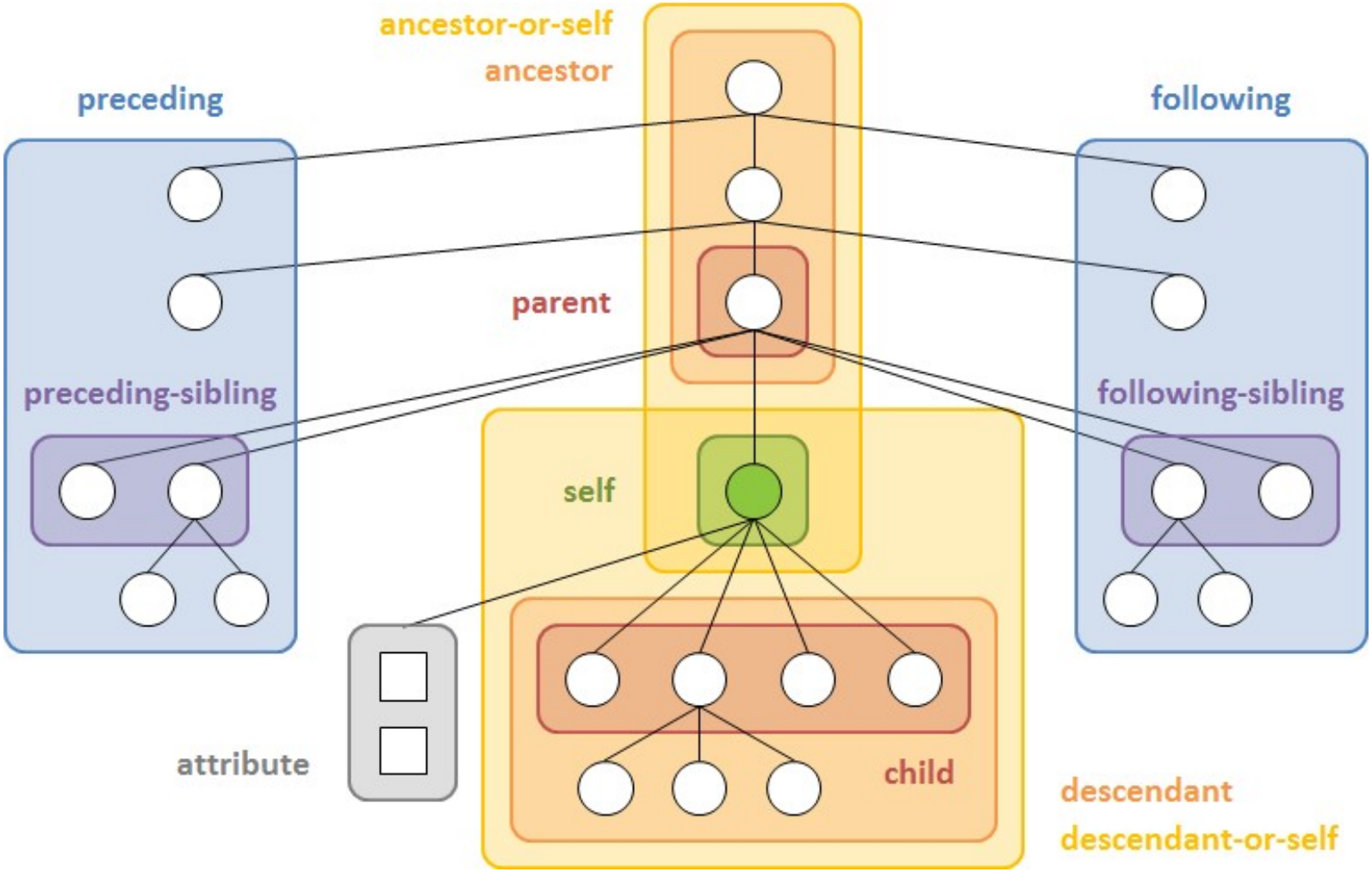
Path Expressions: Axes

Axis

- Selects nodes that are reachable from a given context node



Path Expressions: Axes



Path Expressions: Axes

Forward axes

- self, child, descendant(-or-self), following(-sibling)
- Nodes are returned in the **document order**

Reverse axes

- parent, ancestor(-or-self), preceding(-sibling)
- Nodes are returned in the **reverse document order**

Path Expressions: Axes

Examples

```
/child::movies
```

```
/child::movies/child::movie/child::title/child::text()
```

```
/child::movies/child::movie/attribute::year
```

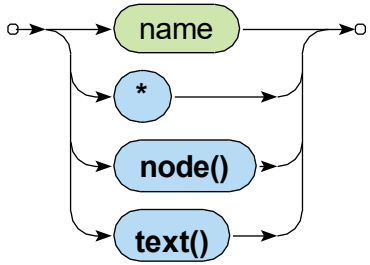
```
/descendant::movie/child::title
```

```
/descendant::movie/child::title/following-sibling::actor
```

Path Expressions: Node Tests

Node test

- Filters the nodes selected by the axis using a basic condition
  - Only **names and kinds** of nodes can be tested



name: elements / attributes with a given name

```
/movies
```

```
/movies/movie/attribute::year
```

Path Expressions: Node Tests

`*`: all elements / attributes

```
/movies/*
```

```
/movies/movie/attribute::*
```

`text()`: all text nodes

```
/movies/movie/title/text()
```

`node()`: all nodes

```
/movies/descendant-or-self::node()/actor
```

Path Expressions: Predicates

Predicates

- Additional filtering of the nodes based on advanced conditions

- When **multiple predicates** are provided…
  - They must all be satisfied
  - They are evaluated one by one, from left to right

Commonly used **conditions**

- Path existence tests, comparisons, position tests
- Logical expressions
- …

Path Expressions: Predicates

Path existence tests

- Relative or absolute path expressions
  - Relative path expressions are evaluated with respect to the node for which a given predicate is tested
- Treated as `true` when evaluated to a **non-empty sequence**

```
/movies/movie[actor]
```

```
/movies/movie[actor]/title/text()
```

Comparisons

- General, value, or node comparison expressions

```
/descendant::movie[@year > 2000]
```

```
/descendant::movie[count(actor) ge 3]/title
```

Path Expressions: Predicates

Position tests

- Allow for filtering of items based on context positions
  - Numbered starting with 1
  - Always relative to the current context (intermediate result)
  - Base order is implied by the axis used

```
/descendant::movie/actor[position() = 1]
```

```
/descendant::movie[actor][position() = last()]
```

Logical expressions

- and, or, not connectives

```
/movies/movie[@year > 2000 and @director]
```

```
/movies/movie[@director][@year > 2000]
```

Path Expressions: Abbreviations

Omitted axis: the default child axis is assumed

```
/movies/movie/title
```

```
/child::movies/child::movie/child::title
```

Attributes: @ ⇔ attribute::

```
/movies/movie/@year
```

```
/movies/movie/attribute::year
```

Descendants: // ⇔ /descendant-or-self::node()/

```
/movies//child::actor
```

```
/movies/descendant-or-self::node()/child::actor
```

Path Expressions: Abbreviations

Context item: . ⇔ self::node()

```
/movies/movie[.//actor]
```

```
/movies/movie[self::node()//actor]
```

Parent: . . ⇔ parent::node()

Position tests: [number] ⇔ [position() = number]

```
/movies/movie/child::actor[2]
```

```
/movies/movie/child::actor[position() = 2]
```

```
/movies/movie[actor][last()]
```

```
/movies/movie[actor][position() = last()]
```

Path Expressions: Conclusion

Evaluation of path expressions

- Evaluated **from left to right, step by step**
  - Result of the entire expression is the **result of the last step**

Only one of the following can be returned…

- Sequence of **nodes**
  - Always sorted in the **document order**
  - **Duplicate nodes are removed**
    - Based on the identities of nodes
- Sequence of **atomic values**
  - The order as well as duplicate values are both preserved

⇒ the returned sequences will never be mixed

Comparison Expressions

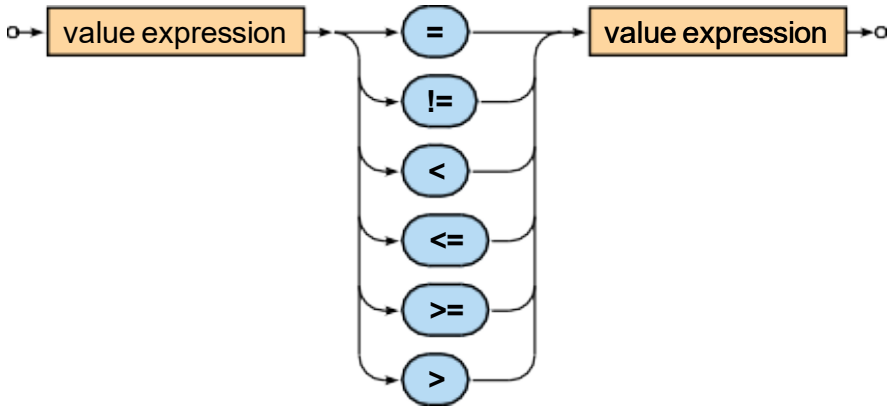
Comparisons

- **General** comparisons
  - Two sequences of values are expected to be compared
  - `=`, `!=`, `<`, `<=`, `>=`, `>`
  - E.g.: `(0,1) = (1,2)`
- **Value** comparisons
  - Two standalone values (singleton sequences) are compared
  - `eq`, `ne`, `lt`, `le`, `ge`, `gt`
  - E.g.: `1 lt 3`
- **Node** comparisons
  - `is` – tests identity of nodes
  - `<<`, `>>` – test positions of nodes (preceding, following)
  - Similar behavior as in the case of value comparisons

Comparison Expressions

General comparisons (**existentially quantified** comparisons)

- Both the operands can be evaluated to **sequences of items** of <u>any length</u>

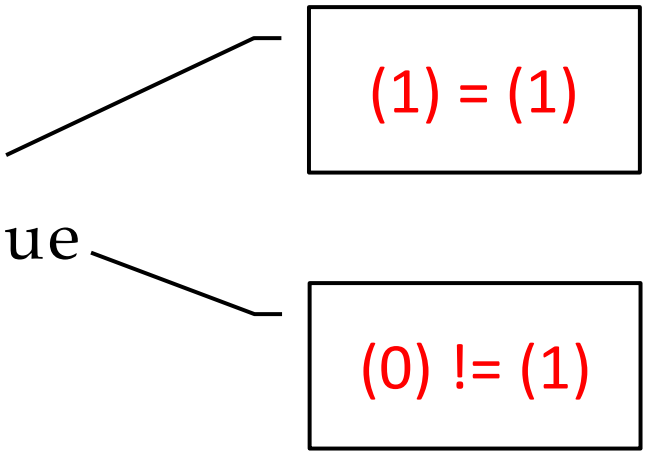


- The result is `true` if and only if **there exists at least one pair of individual items** satisfying a given relationship

Comparison Expressions: Examples

General comparisons

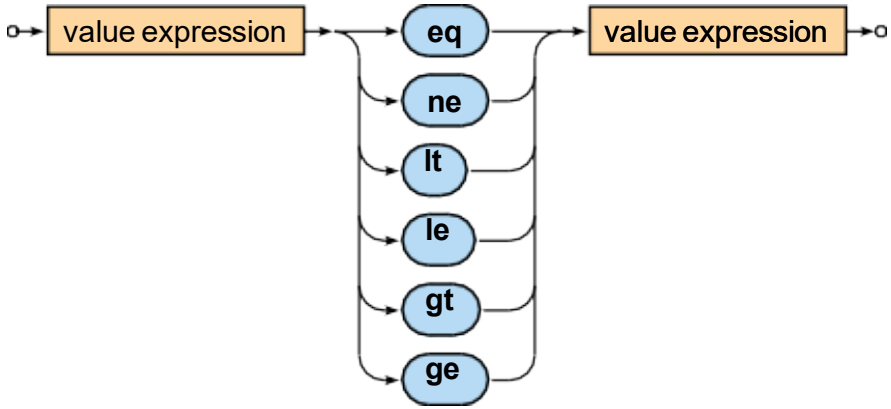
- `(1) < (2)` = true
- `(1) < (1,2)` = true
- `(1) < ()` = false
- `(0,1) = (1,2)` = true
- `(0,1) != (1,2)` = true



Comparison Expressions

Value comparisons

- Both the operands must be evaluated to **singleton sequences**



- **Empty sequence** ( ) is returned…
  - when at least one operand is evaluated to an empty sequence
- **Type error** is raised…
  - when at least one operand is evaluated to a longer sequence

Comparison Expressions: Examples

Value comparisons

- `(1) le (2)` = true
- `(1) le ()` = ()
- `(1) le (1,2)` ⇒ error
- `() le (1,2)` = ()

Comparison Expressions

Value and general comparisons

- **Atomization of values** – applied automatically
  - Atomic values are preserved untouched
  - **Nodes are transformed to atomic values**
- In particular…
  - **Element** node is transformed to a string with concatenated text values it contains in the document order
    - E.g.: <movie year="2006">Vratné lahve</movie> is atomized to a string Vratné lahve
    - I.e., attribute values and element names are not included!
  - **Attribute** node is transformed to its value
  - **Text** node is transformed to its value

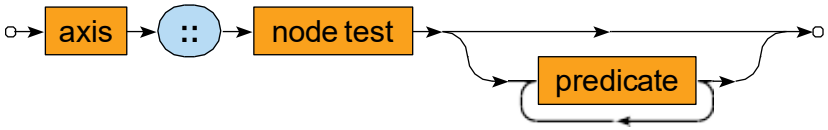
Comparison Expressions: Examples

Value and general comparisons

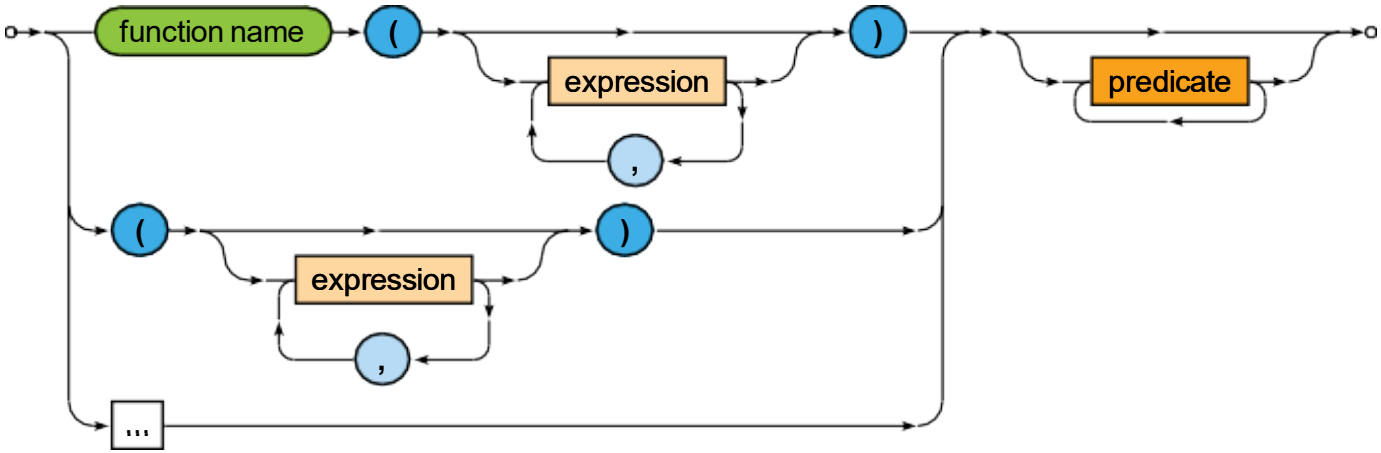
- `<a>5</a> eq <b>5</b>` = true
- `<a>12</a> = <a><b>1</b>2</a>` = true
- `<a t="1">3</a> lt 5` = true

Path Operator

Navigational steps in path expressions



- Extended functionality (XPath 2.0)
  - **Function calls**, **sequence constructors**, …
  - Must not yield mixed sequences (nodes and atomic values)



Path Operator: Examples

Numbers of actors who appeared in the individual movies

```
/movies/movie/count(actor)
```

```
2
3
2
```

Flat sequence of interleaved movie titles and years of filming

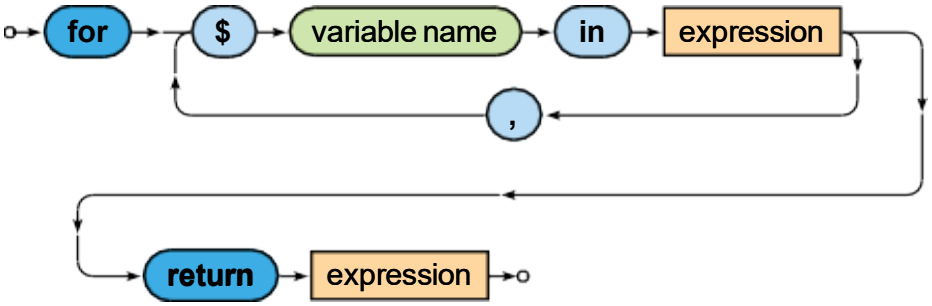
```
/movies/movie/(title, @year)/data(.)
```

```
2006
Vratné lahve
2000
Samotáři
2007
Medvídek
```

Iteration Expressions

For expression (XPath 2.0, XQuery 1.0 FLWOR)

- Allows for the **iteration** over **items** of an input sequence / **tuples of items** when more input sequences are provided



- Returns a flat sequence containing results of the `return` clause evaluated for each input item / tuple of items

Iteration Expressions: Example

Numbers of actors who appeared in movies filmed in *2000* or later

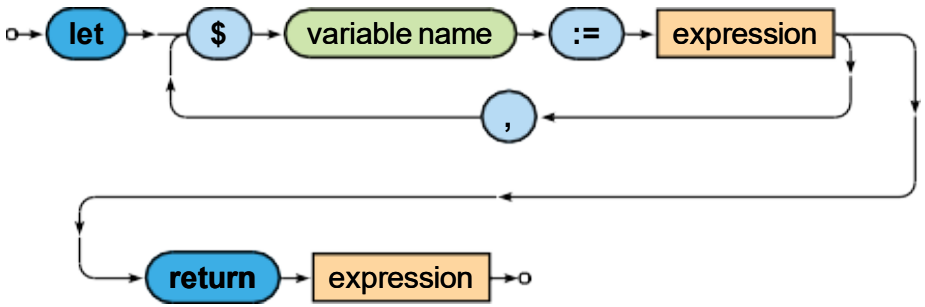
```
for $m in //movie[@year >= 2000]
return count($m/actor)
```

```
2
3
2
```

Variable Assignments

Let expression (XPath 2.0, XQuery 1.0 FLWOR)

- Allows for **assignment** of one or more **variables**
  - $ is used to denote variables
- These variables can then be accessed later on



- Returns the result of the evaluated `return` clause

Variable Assignments: Example

Find titles of movies with at least average rating

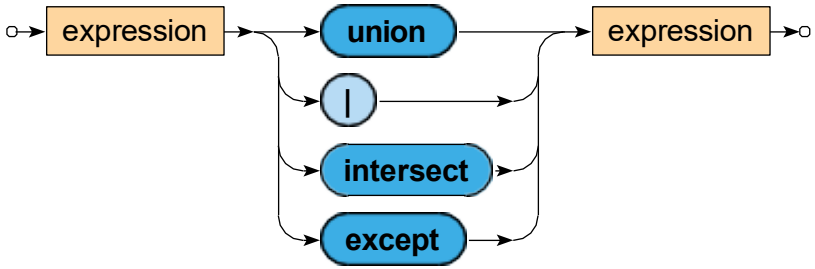
```
let $a := avg(//movie/@rating)
return //movie[@rating >= $a]/title/text()
```

```
Vratné lahve
Samotáři
```

Set Operations

Traditional **set operations** (XPath 2.0)

- Only applicable on sequences of nodes (not atomic values)!
- Duplicate nodes are removed



Union

- Nodes that occur in either of the operands

Intersection

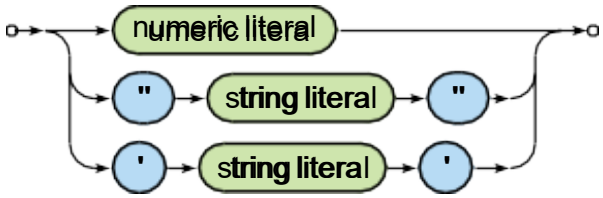
- Nodes that occur in both the operands

Difference

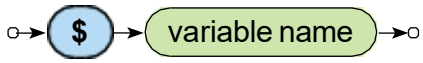
- Nodes that occur in the first operand but not in the second one

Primary and Other Expressions

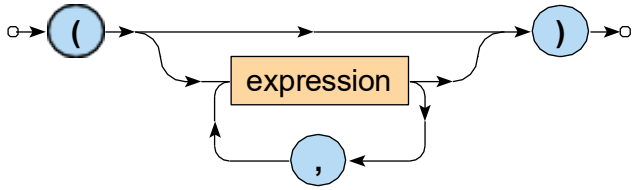
Literals



Variable reference

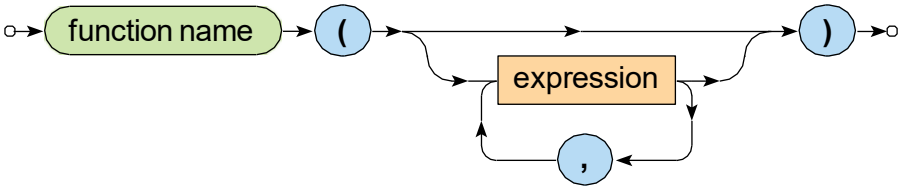


Sequence constructor



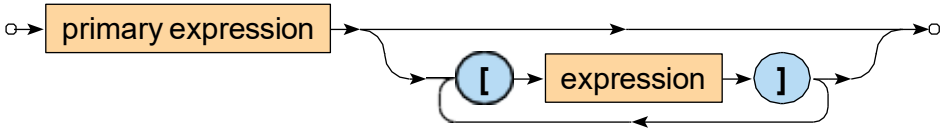
Primary and Other Expressions

Function call



Postfix expression

- Allows to add **predicates** to primary expressions
  - I.e., **variable references**, **sequence constructors**, …



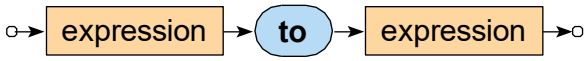

```
$movies[actor]
```

```
(//actor)[last()]
```

Primary and Other Expressions

Range expression

- Allows to generate a **sequence of consecutive integers**




```
/movies/movie[@year = 2011 to 2020]
```

Arithmetic expressions

Conditional expression

Quantified expressions

- Allow to simulate the **existential and universal quantifiers**

Predefined Functions

`data($items)`

- Performs the **atomization of values** in a sequence

`avg($items)`, `sum($items)`, `min($items)`, `max($items)`

- Calculates the **average** / **sum** / **minimum** / **maximum**

`name($node)`

- Returns the **name of an element or attribute** node

`position()` and `last()`

- Returns the current **context position** / **context size**

`string-join($items, $separator)`

- Returns a **string with concatenated atomized values**

`doc($uri)`

- Returns the **document node** for a specified XML file

Lecture Conclusion

XPath

- **Path expressions**
  - Absolute and relative paths
  - Axes, node tests, and predicates
- **Comparison expressions**
  - General, value, and node comparisons
- For and let expressions
- Set operations
  - Union, intersection, difference
- Primary expressions
- …