XML and Databases

Chapter 10: XPath I: Location Paths

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Winter 2022/23

http://www.informatik.uni-halle.de/~brass/xml22/

Objectives

After completing this chapter, you should be able to:

- write XPath expressions for a given application.
- explain what is the result of a given XPath expression with respect to a given XML data file.
- explain how comparisons are done, and why XPath has two sets of comparison operators (e.g. `=` vs. `eq`).
- define “atomization”, “effective boolean value”.
- enumerate some axes and explain abbreviations.
- explain features needed for static type checking.

Contents

Introduction (1)

- XPath ("XML Path Language") is a standard for expressions that are mainly used for selecting parts of XML documents (nodes in the XDM tree).

  One can view this important subset of XPath as a pattern language: A tree node matches a pattern if it is contained in the result of evaluating the XPath expression (a sequence of nodes).

- However, XPath expressions can also compute atomic values or more generally any sequence allowed by XDM.

  Arithmetic expressions map numbers to numbers, XPath maps a set of documents (or really a "context", see below) to a sequence of nodes and atomic values. So it does not seem to be closed.

Introduction (2)

- XPath is used e.g. in
  - XSLT (XML Stylesheet Lang./Transformations)

    E.g. for defining to which nodes a transformation template should be applied, which parts of the input document should be copied to the output document, and where processing in the input document should continue after a template was applied.
  - XPointer

    To permit references to a part of a document. With classic URIs (plus "`#...`"), one can point only to places in an HTML document, where the author of the document has placed an anchor.
  - XML Schema

    For selecting nodes that are uniquely identified etc.
  - XQuery (XML Query Language)

Introduction (3)

- The reason for the name "XPath" is that the expressions are quite similar to path expressions in e.g. the UNIX file system (directory tree).

  However, XPath expressions are actually much more powerful. One could imagine a future operating system that uses an XDM tree (or something similar) to replace its file system.

- For example,

  ```
  /GRADES-DB/STUDENTS/STUDENT
  ```

  is an XPath-expression that selects `STUDENT`-nodes that are children of (the) `STUDENTS` node that is a child of the `GRADES-DB` document element.

Introduction (4)

- Path expressions are used also in object-oriented languages for navigating in complex structures.

  E.g., in OQL. Again, they are much simpler than XPath. By the way, there a full stop "." is used instead of "/". The relational model does not need path expressions because of its simple (flat) structure.

- One can view XPath as a simple query language for XML.

  It does not have joins and aggregations, but it has quite powerful selections, and it has certian forms of semi-joins.

- XPath has not itself XML syntax.

  This is more compact. Furthermore, XPath is used in attributes.

XPath Standards (1)

- XPath 1.0 is a W3C Recommendation since 16 November 1999.

  [https://www.w3.org/TR/1999/REC-xpath-19991116/]

  It began with work on the XSL Pattern Language, and the "location paths" in drafts of the XPointer specification. XPath unified the two.

  The first W3C Recommodation for XML was published in February 1998.

- XPath 2.0 was published as W3C Recommendation on 23 January 2007 (Second Edition on 14 December 2010).

  [https://www.w3.org/TR/xpath20/]

  The main change from XPath 1.0 is the stricter typing. In 1999, when XPath 1.0 was published, there was no XML Schema yet (work on XML Schema had just begun, XML Schema 1.0 was published in May 2001). XPath 2.0 uses XML Schema types. Furthermore, variable bindings and nested subqueries were added. XPath 2.0 has a compatibility mode that removes most (but not all) incompatibilities with XPath 1.0.

XPath Standards (2)

- XML Path Language (XPath) 3.1 was published on 21 March 2017.

  [https://www.w3.org/TR/xpath-31/]

  The main change is the support for maps and arrays in order to make XPath applicable not only for XML, but also for JSON. In comparison to the large step from XPath 1.0 to XPath 2.0, the changes are not very big. There was also a version 3.0 published on 08 April 2014 (however, Maps and Arrays where added only in Version 3.1)

  [https://www.w3.org/TR/xpath-30/]

- The version explained on the following slides is 2.0.

  There might be a later chapter about JSON.

- A function library was specified in a separate standard (see Chapter 12).

  [https://www.w3.org/TR/xquery-operators/]

Contents

Software (1)

- Free Formatter Webpage

  [http://www.freeformatter.com/xpath-tester.html]

- XLab: Online XPath experiments

  [http://www.zvon.org:9001/saxon/cgi-bin/XLab/XML/xlabIndex.html?stylesheetFile=XSLT/xlabIndex.xslt]

- One can write a simple XSLT stylesheet that shows the result of an XPath expression. Then any XSLT processor (e.g., in a web browser) can be used. Often only XPath 1.0!

  How to do this is shown below. Also links to XSLT processors are given that are independent of a browser (might give better error checking).

- An XPath expression is already a simple XQuery query. Thus, any XQuery processor can be used.

  XQuery implementations are listed below (some with online demo).

Software (2)

XQuery Implementations:

- BaseX

  [https://basex.org/]

- Galax

  Open source, from some authors/editors of the XQuery Specification.
  [http://www.galaxquery.org/]

- X-HIVE

  Commercial XML-DBMS, Online demo evaluator.
  [http://support.x-hive.com/xquery/].

- AltovaXML

  The engine used in XMLSpy is free (contains validator: DTD/Schema, XSLT 1.0/2.0, XQuery). [http://www.altova.com/altovaxml.html]

Software (3)

XQuery Implementations, continued:

- Qizx/open (open source Java implementation)

  [http://www.axyana.com/qizxopen/] Online demonstration:
  [http://www.xmlmind.com:8080/xqdemo/xquery.html]

- Saxon (from Michael Kay)

  Michael Kay is editor of the XSLT 2.0 specification. The basic version of Saxon (without static type checking and XQuery→Java compiler) is open source. It includes support for XSLT 2.0, XPath 2.0 and XQuery 1.0.
  [http://saxon.sourceforge.net/]

- eXist (open source native XML database)

  [http://exist.sourceforge.net/]
  Online demo: [http://demo.exist-db.org/sandbox/sandbox.xql]

Software (4)

XSLT Implementations:

- Any modern web browser has XSLT support.

  See, e.g., `http://www.mozilla.org/projects/xslt/`.

- Xalan (Apache)

  [http://xalan.apache.org/]

- XT (James Clark)

  [http://www.blnz.com/xt/index.html], [http://www.jclark.com]

- Sablotron

  [http://www.gingerall.org/sablotron.html]

- See above: Saxon, AltovaXML.

Trying XPath with XSLT (1)

- Modern web browsers can apply an XSLT stylesheet to visualize XML (by transforming it to HTML).
- Thus, one writes a reference to the stylesheet in the XML data file (input for XPath query), e.g.:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
        href="query.xsl"?>
<GRADES-DB>
    ...
</GRADES-DB>
```

Trying XPath with XSLT (2)

- Then one looks at this data file in the browser.
  It automatically fetches the stylesheet `query.xsl`
  (see next four slides) and uses it for the transformation.

  [https://users.informatik.uni-halle.de/~brass/xml22/examples/ex2_query.xml]

- The stylesheet file mainly contains a transformation rule that evaluates an XPath expression (with the root node as starting point) and only shows the result of this expression in the output.

  At least in Firefox, one cannot do this with local files: The "The Same Origin Policy" prevents accessing the Stylesheet. One must use `http:/https:`.

- However, additional transformation rules are necessary to format the result of the XPath expression (arbitrary XDM nodes) as HTML.

  I do not know how to handle the document node. Hints welcome.

Trying XPath with XSLT (3)

```
<?xml version="1.0"?>

<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">

  <xsl:output method="html"
      encoding="ISO-8859-1"
      doctype-public="-//W3C//DTD HTML 3.2 Final//EN"
      indent="yes"/>
```

Trying XPath with XSLT (4)

```
<xsl:template match="/">
    <html>
    <head><title>Query Result</title></head>
    <body>
    <ul>
        <xsl:apply-templates
            select="//STUDENT/LAST"/>
            <!-- This is the XPath expression
                 to be tested -->
    </ul>
    </body>
    </html>
</xsl:template>
```

Trying XPath with XSLT (5)

```
<xsl:template match="*">
   <li>ELEMENT: <xsl:value-of select="name(.)"/>
                (<xsl:value-of select="."/>)</li>
</xsl:template>

<xsl:template match="@*">
   <li>ATTRIBUTE: <xsl:value-of select="name(.)"/>
                  (<xsl:value-of select="."/>)</li>
</xsl:template>

<xsl:template match="text()">
    <li>TEXT: <xsl:value-of select="."/></li>
</xsl:template>
```

Trying XPath with XSLT (6)

```
<xsl:template match="comment()">
    <li>COMMENT: <xsl:value-of select="."/></li>
</xsl:template>

<-- xsl:template match="/">
   <li>DOCUMENT: <xsl:value-of select="."/></li>
</xsl:template -->

<xsl:template match="processing-instruction()">
   <li>PROC-INSTR: <xsl:value-of select="name(.)"/>
                (<xsl:value-of select="."/>)</li>
</xsl:template>

</xsl:stylesheet>
```

Contents

Context (1)

- An expression is evaluated relative to a context.
- In XPath 1.0, the context consisted of:
  - a node (context node)
  - a context position (position of context node in current set/sequence: positive integer 1, 2, ...)
  - a context size (number of nodes in current set: positive integer)
  - a set of variable bindings
  - a function library
  - a set of namespace declarations

Context (2)

- XPath 2.0 distinguishes static and dynamic context of an expression.
- The reason is that XPath expressions can possibly be compiled and optimized, and afterwards executed many times on different documents.
- In this phase, also static type checking is done.
- The actual (dynamic) types of the values that are computed during evaluation of an expression are either equal to the static type of the expression or more specific (derived from the static type).

Context (3)

- Dynamic context:
  - context item (atomic value or node)
  - context position
  - context size
  - variable values
  - function implementations
  - current dateTime
  - implicit timezone
  - available documents
  - available collections, default collection

Context (4)

- Remarks about dynamic context:
  - If the context item is a node, it is called context node.
  - Context item, context position and context size are together called the focus of an expression.
  - The current dateTime is used for the XPath function `current-dateTime`.

    It is guaranteed that if this function is accessed multiple times during an evaluation of an expression, it always returns the same value. This simplifies optimizations.

Context (5)

- Remarks about dynamic context, continued:
  - The implicit timezone is used for `dateTime`-values without timezone ("local time") when comparing them with values with timezone (UTC).

    This seems not quite compatible with the XML Schema specification which treats values in local time as if they could possibly be in any timezone, leading to a partial order.
  - Available documents and collections are used for the functions `doc` and `collection`.

    `doc` maps a URI to a document node, and `collection` maps a URI to a sequence of nodes. The function `collection` can also be called without argument, then it returns the default collection.

Context (6)

- An important part of the static context is type information.
- XPath is always used embedded in another language (e.g. XSLT, XQuery).
- There are many parameters that are needed for evaluating an XPath expression that must somehow be set in the host language (e.g., namespaces).

  Also collations are needed for string comparisons.
- These are also part of the static context.

Context (7)

- Static context:
  - XPath 1.0 compatibility mode.

    This is true when the XSLT version is not 2.0.
  - Statically known namespaces.

    I.e. the namespace prefixes declared for the XPath expression.
  - Default namespace for element and data types.

    In XSLT, this can be set with `xsl:xpath-default-namespace="URI"`.
  - Default namespace for functions.

    XPath functions are in `http://www.w3.org/2005/xpath-functions`. XSLT automatically initializes this component of the static context with the standard namespace, so no prefix is needed when calling XPath functions.

Context (8)

- Static context, continued:
  - Schema information (types/elements/attributes)
  - Variable declarations (name and type).
  - Static type of context item.
  - Function signatures (name, input/result types)
  - Known collations, default collation.
  - Base URI.
  - Statically known documents/collections.

    The default type for a call to `document` is `document-node()?`, and for `collection`, it is `node()*`. If information should be available already during compilation, the types could be different (more specific?).

Contents

Location Paths (1)

- The purpose of an location path (or "path expression") is to select nodes in an XDM tree.

  Actually, in its very last step, it can also compute a sequence of atomic values (or a single value), not only a sequence of nodes.
  A path expression is not the most general kind of XPath expression, but it is the kind that is most often used.

- There are absolute and relative paths:
  - An absolute path starts with a "/" or "//", followed by a relative path.

    For "/", the relative path is optional. For "//", it is required.
  - A relative path consists of a series of steps, separated by "/" or "//".

Location Paths (2)

- The "//" will later be explained as an abbreviation:
  - The syntax must be defined including all abbreviations.
  - For the semantics, it suffices to treat only XPath expressions, in which the abbreviations are fully expanded (normalized expressions that do not contain e.g. "//").
- A step can be
  - an axis step (in full or abbreviated syntax),
  - a filter expression.

Location Paths (3)

- An axis step in full (verbose) syntax has the form

  axis::node-test[predicate]*

  The * means that the predicate part may be missing or may be repeated. "axis", "node-test" and "predicate" are explained below (nonterminals of the grammar), "`::`", "`[`", and "`]`" must be written as given (terminal symbols of the grammar).

- The axis (e.g., `child`) selects a sequence of nodes by their position relative to the context node.
- The node test selects a subset of these nodes by their name or type (kind).
- The predicate(s) contain further conditions on the resulting nodes (e.g., position, value).

Location Paths (4)

- A filter expression consists of a primary expression followed by a sequence of zero or more predicates in "`[...]`".
- A primary expression is:
  - Any XPath expression in parentheses `(...)`.
  - A data type literal (constant), e.g. `"abc"`.
  - A function call.
  - A variable reference, e.g. `$x`.
  - A context item reference: "`.`"

Location Paths (5)

- `E1/E2` is evaluated as follows:
  - `E1` is evaluated. The result must be a (possibly empty) sequence of nodes, otherwise a type error is raised.
  - `E2` is evaluated once for every node in the result of `E1` as context node.

    The context size is the length of the result of `E1`. The context position is the position of the context node in the sequence (depending on the axis, the position might be counted from the end of the sequence, see below).

Location Paths (6)

- Evaluation of `E1/E2`, continued:
  - If each evaluation of `E2` returns a sequence of nodes, the result of `E1/E2` is the union of the nodes in these sequences in document order (with duplicates removed).
  - If each evaluation of `E2` returns a sequence of atomic values, these sequences are concatenated (without duplicate elimination).
  - If `E2` returns nodes and atomic values, a type error is raised.

Contents

XPath Axis (1)

- An axis selects a sequence of nodes based on their position in the document tree relative to the current context node.
- There are 13 axis (in XPath 1.0 and in XPath 2.0).
- Of these, 8 are forward axes (cont. on next page):
  - `self`
  - `child`
  - `descendant`
  - `descendant-or-self`
  - `following-sibling`

XPath Axis (2)

- Forward axes, continued:
  - `following`
  - `attribute`
  - `namespace` (deprecated, not in XQuery)
- There are 5 reverse axes:
  - `parent`
  - `ancestor`
  - `ancestor-or-self`
  - `preceding-sibling`
  - `preceding`

XPath Axis (3)

- A minimal XPath implementation needs to support only the following axes:
  - `self`
  - `child`
  - `parent`
  - `descendant`
  - `descendant-or-self`
  - `attribute`

XPath Axis (4)

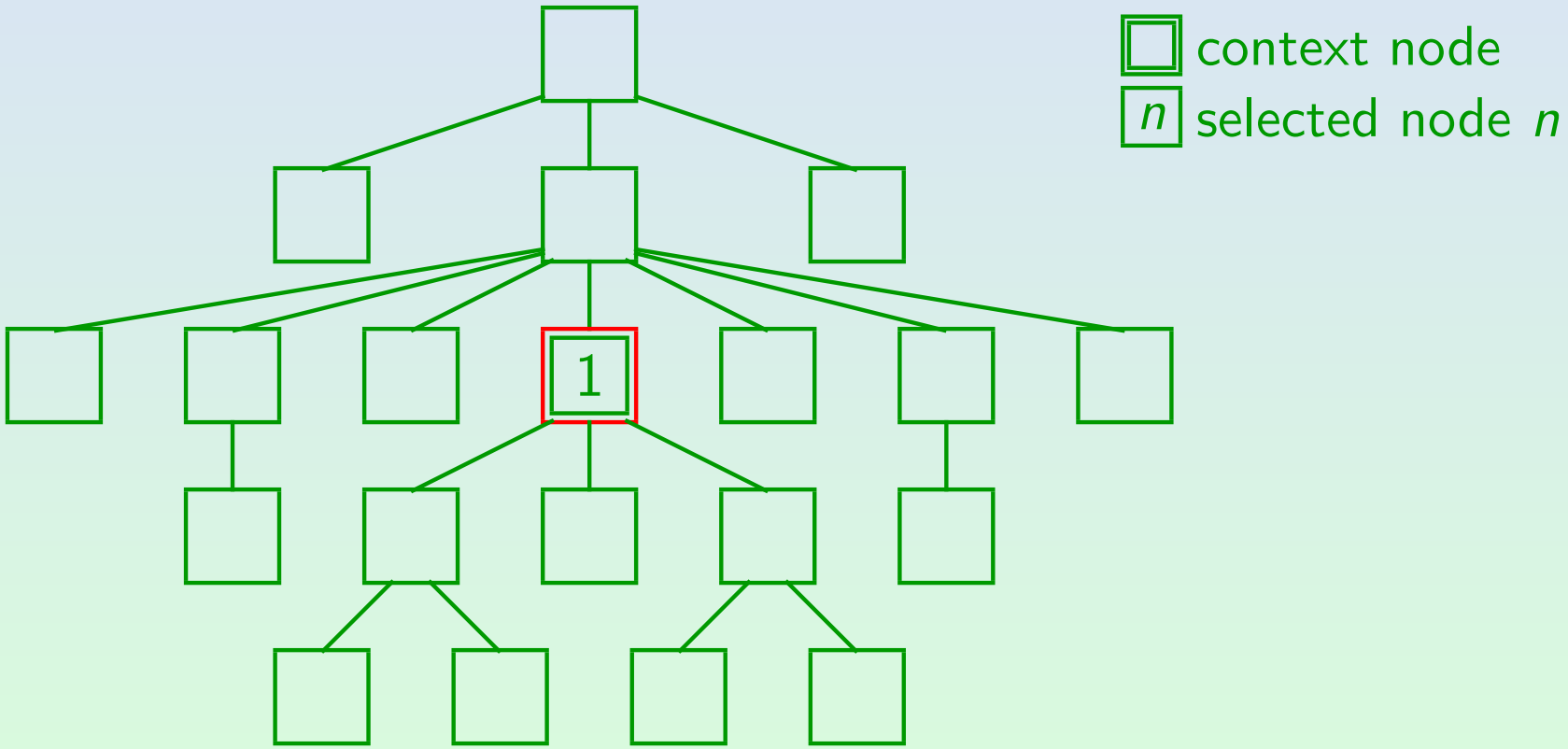
- The following axes partition a document (except attribute and namespace nodes): `self`, `ancestor`, `descendant`, `preceding`, `following`.
- If an axis is a reverse axis, the context position used for evaluating predicates in this location step is assigned in inverse document order.

  For forward axes, it is assigned in document order. If the predicate is not in a location step, the position is the position in the sequence.
- The selected nodes with their position are shown in an example on the following slides.

  The context node is marked with a double border.

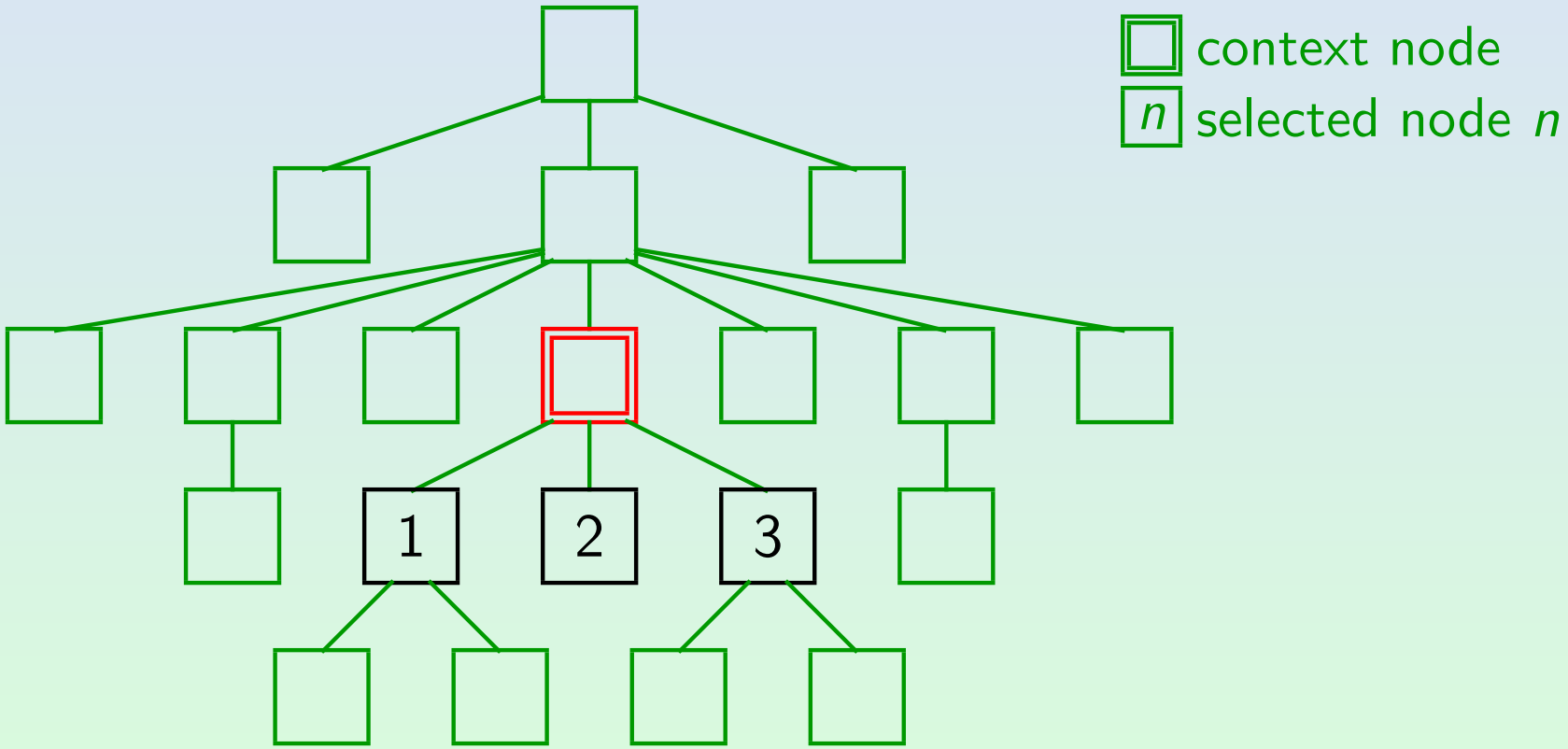
XPath Axis (5)

`self:`



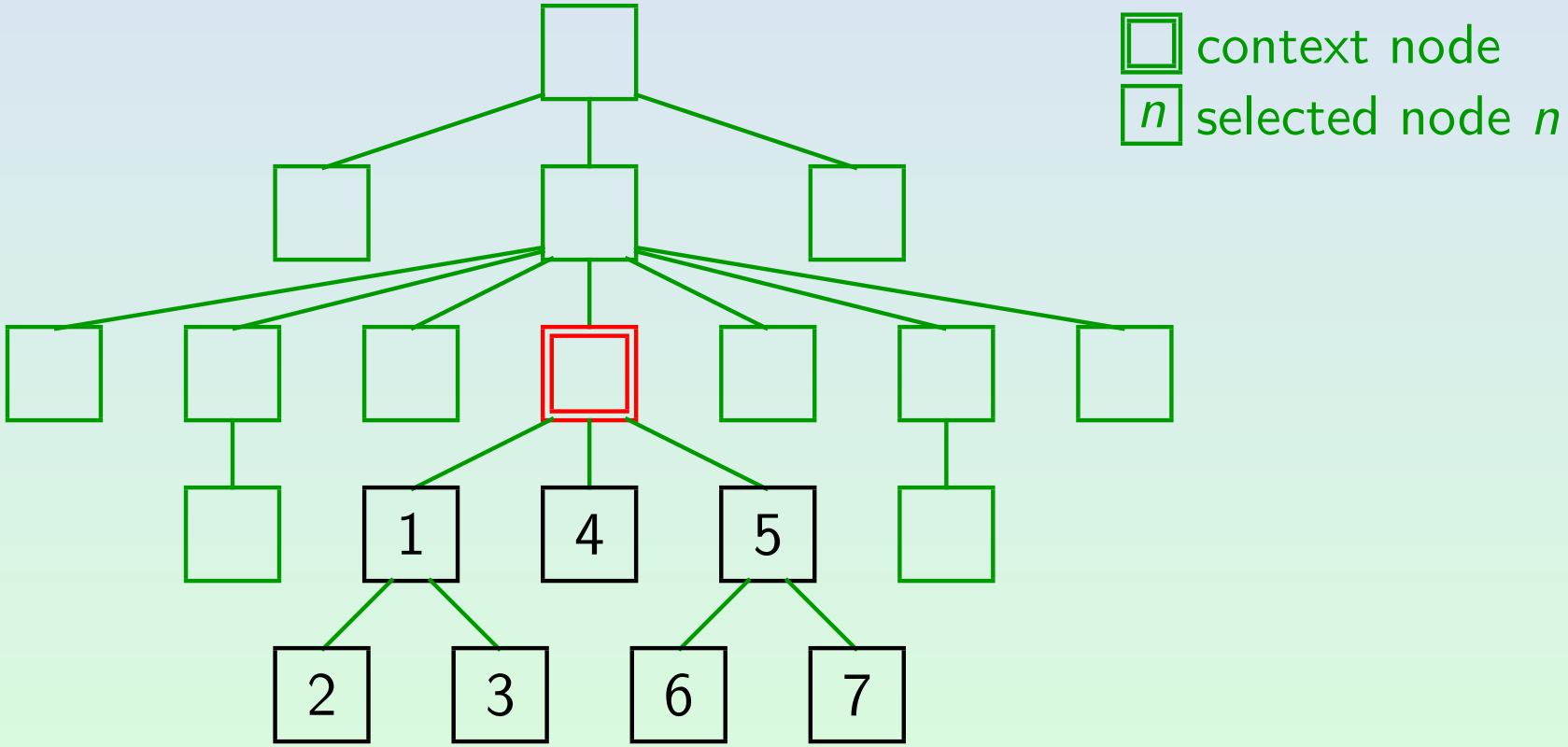
XPath Axis (6)

`child`:



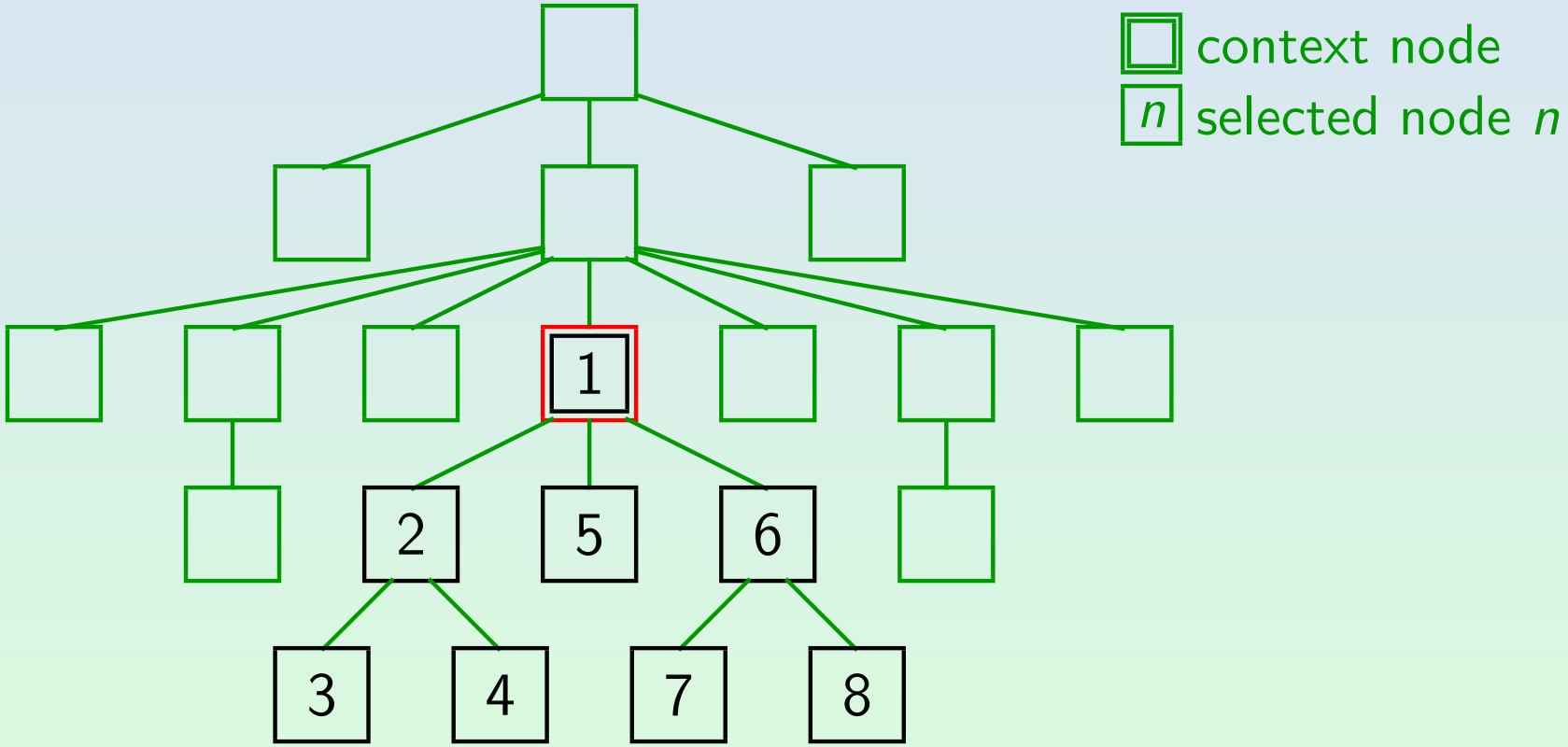
XPath Axis (7)

`descendant:`



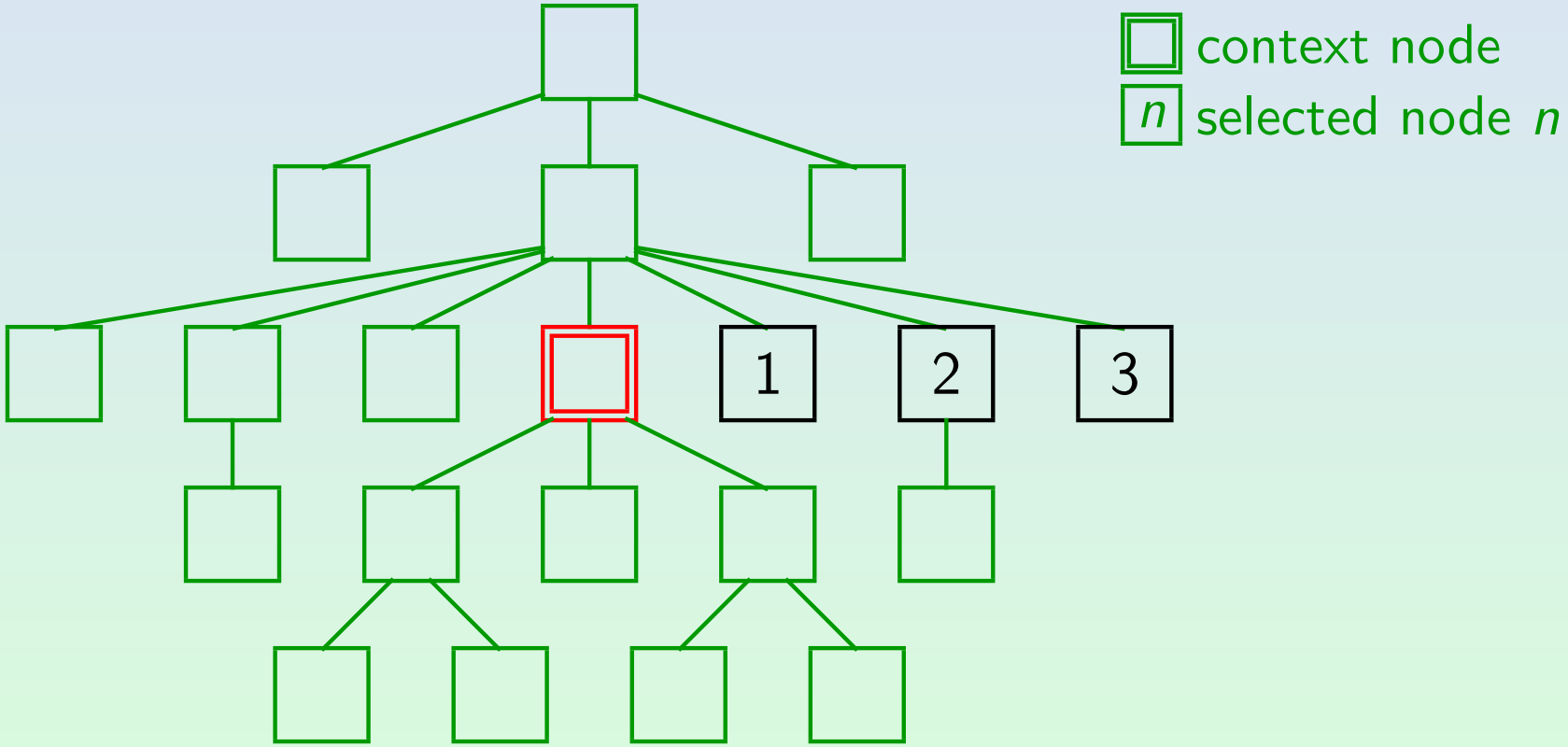
XPath Axis (8)

`descendant-or-self:`



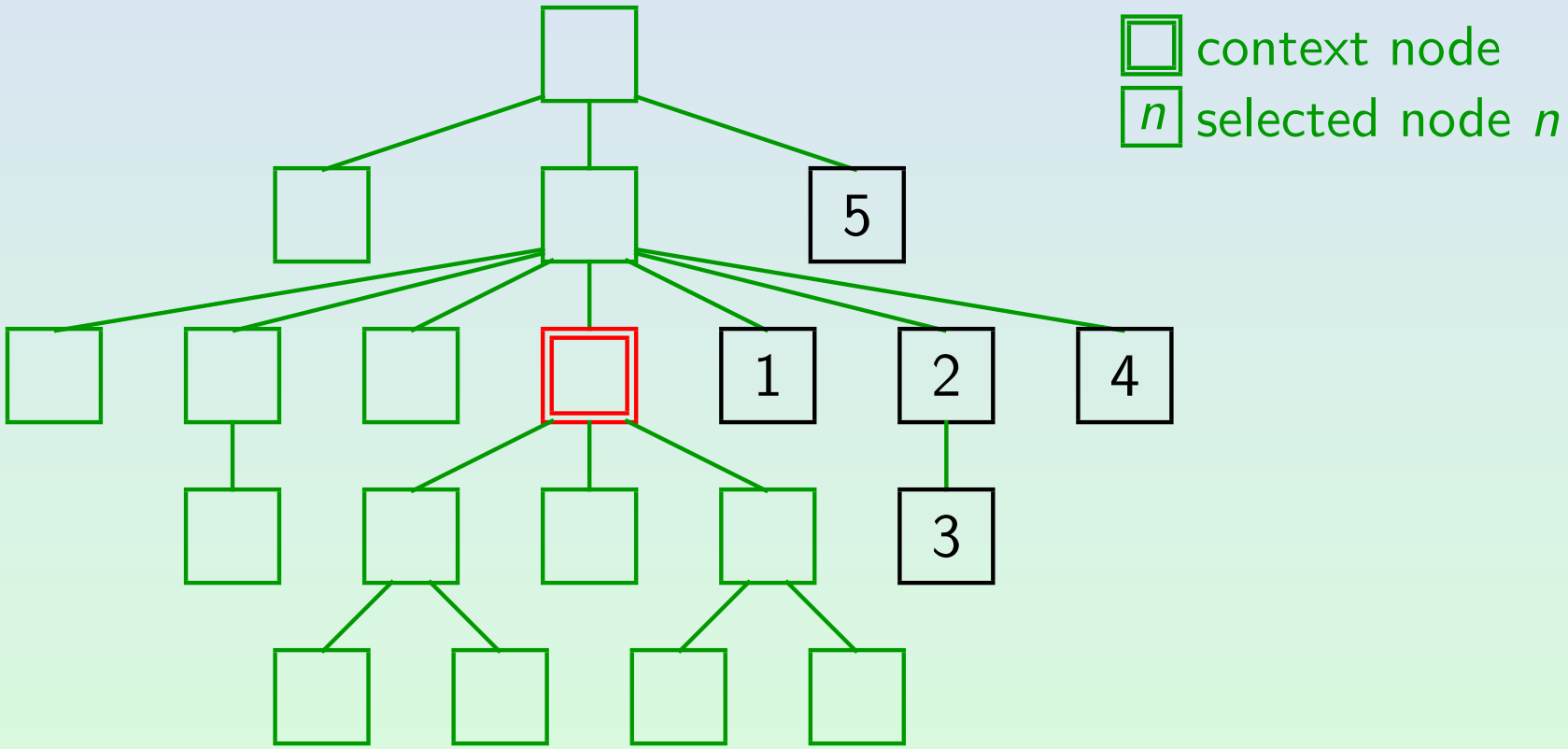
XPath Axis (9)

following-sibling:



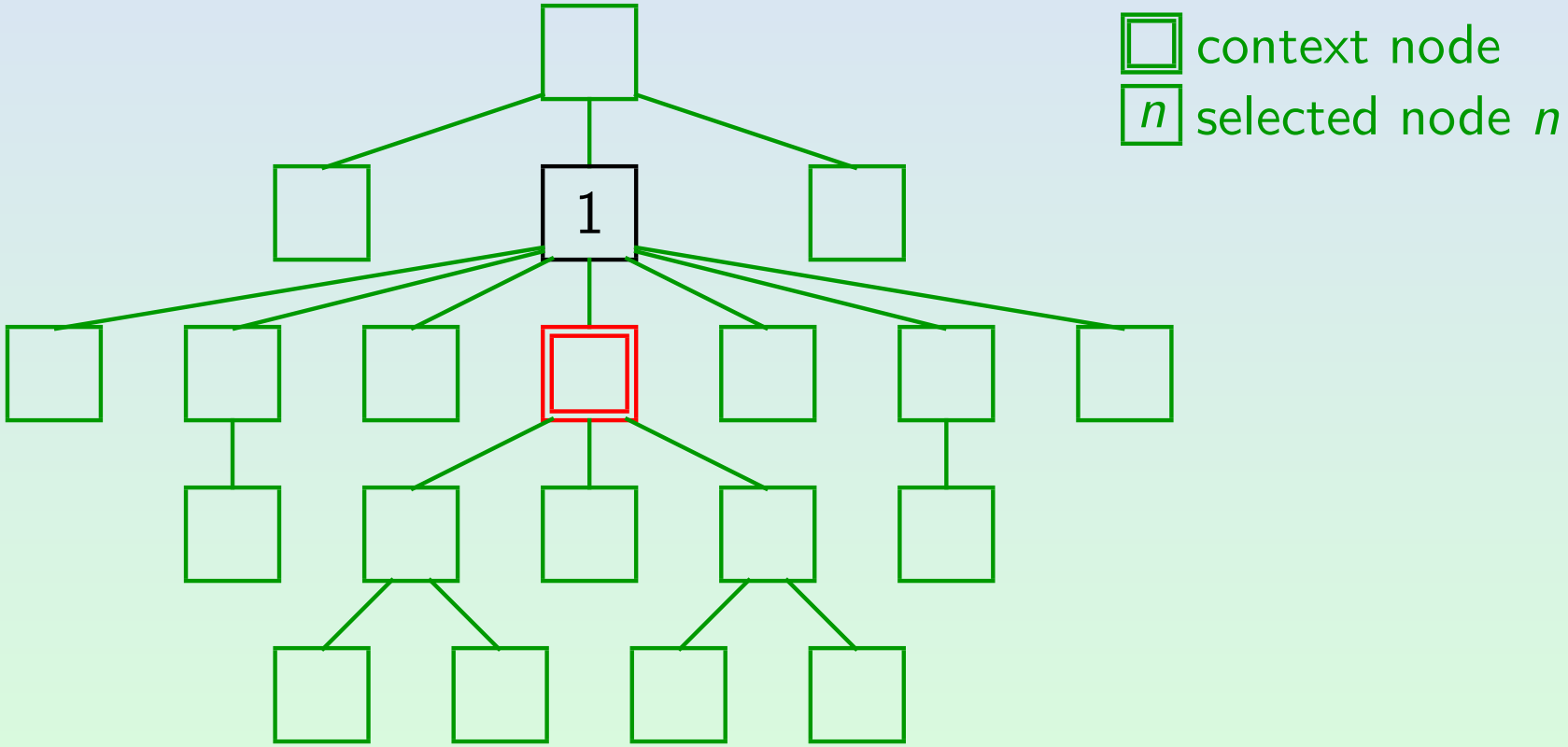
XPath Axis (10)

following:



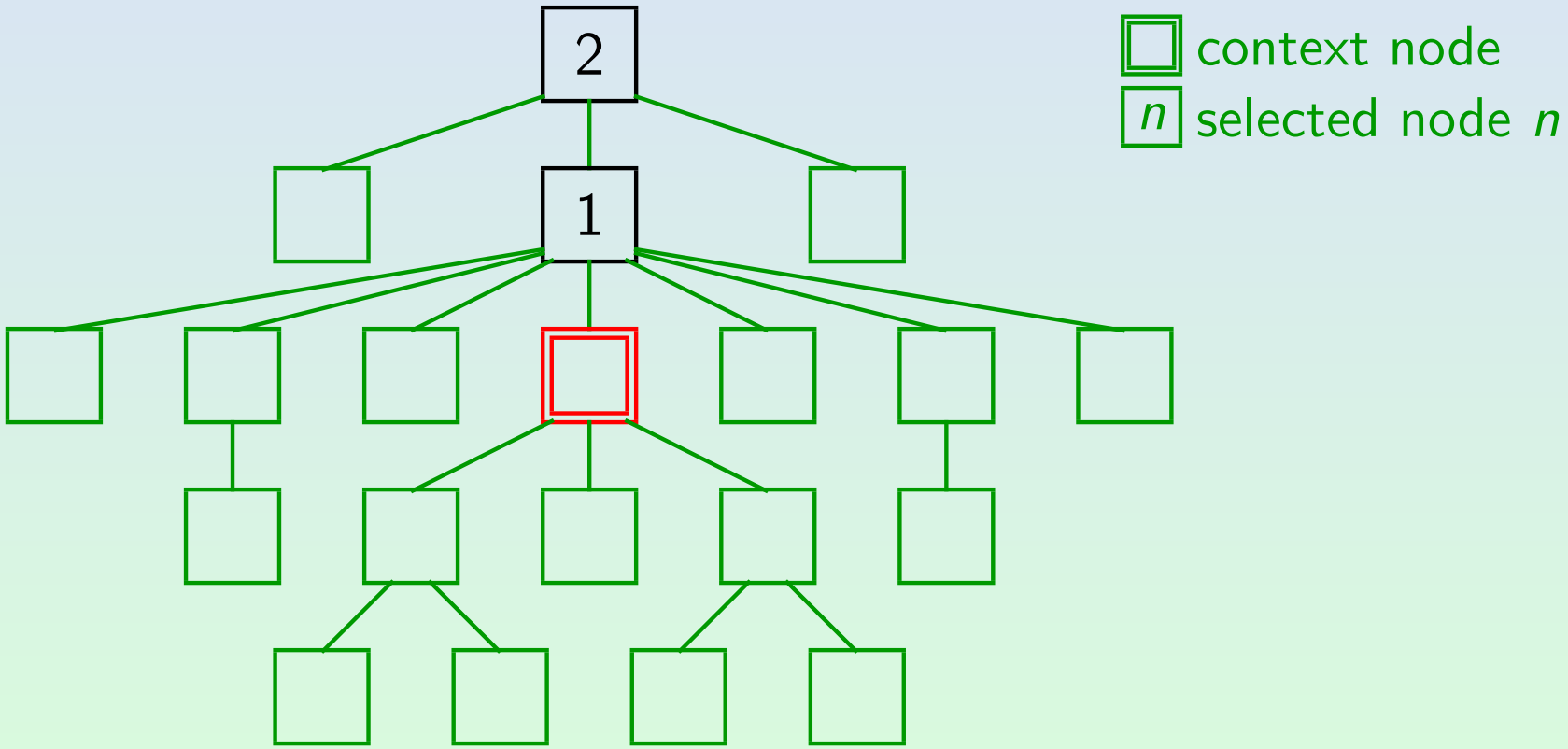
XPath Axis (11)

parent:



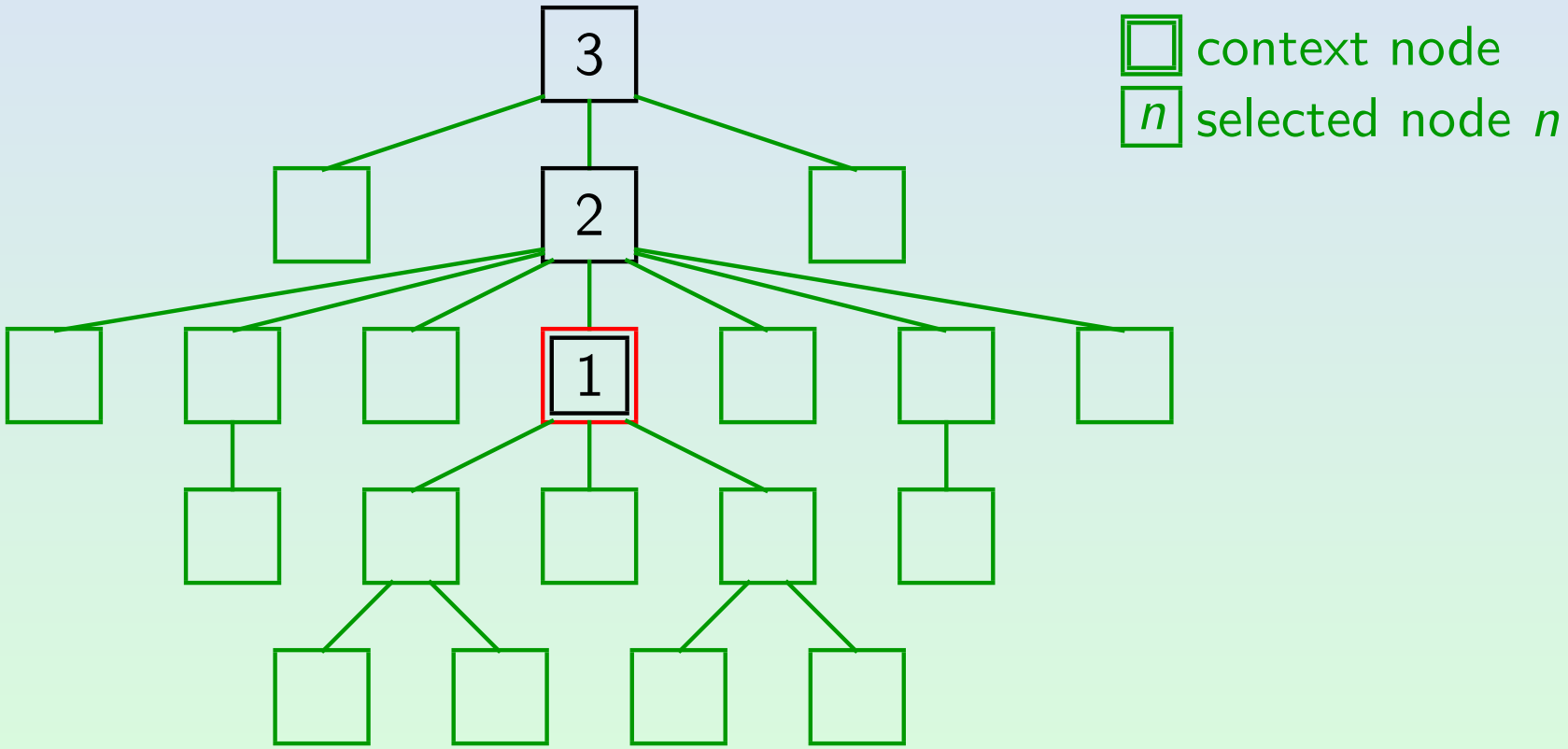
XPath Axis (12)

ancestor:



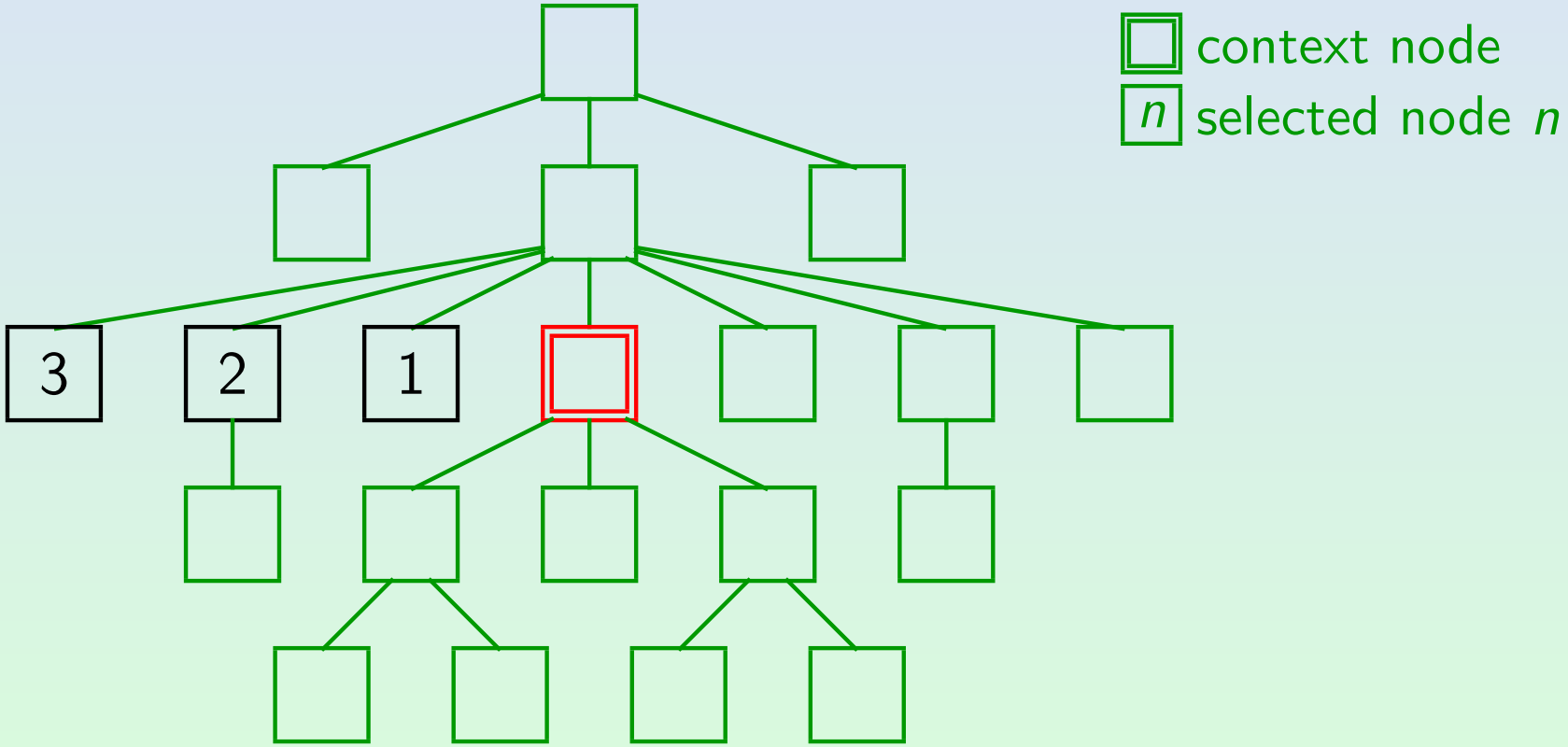
XPath Axis (13)

`ancestor-or-self:`



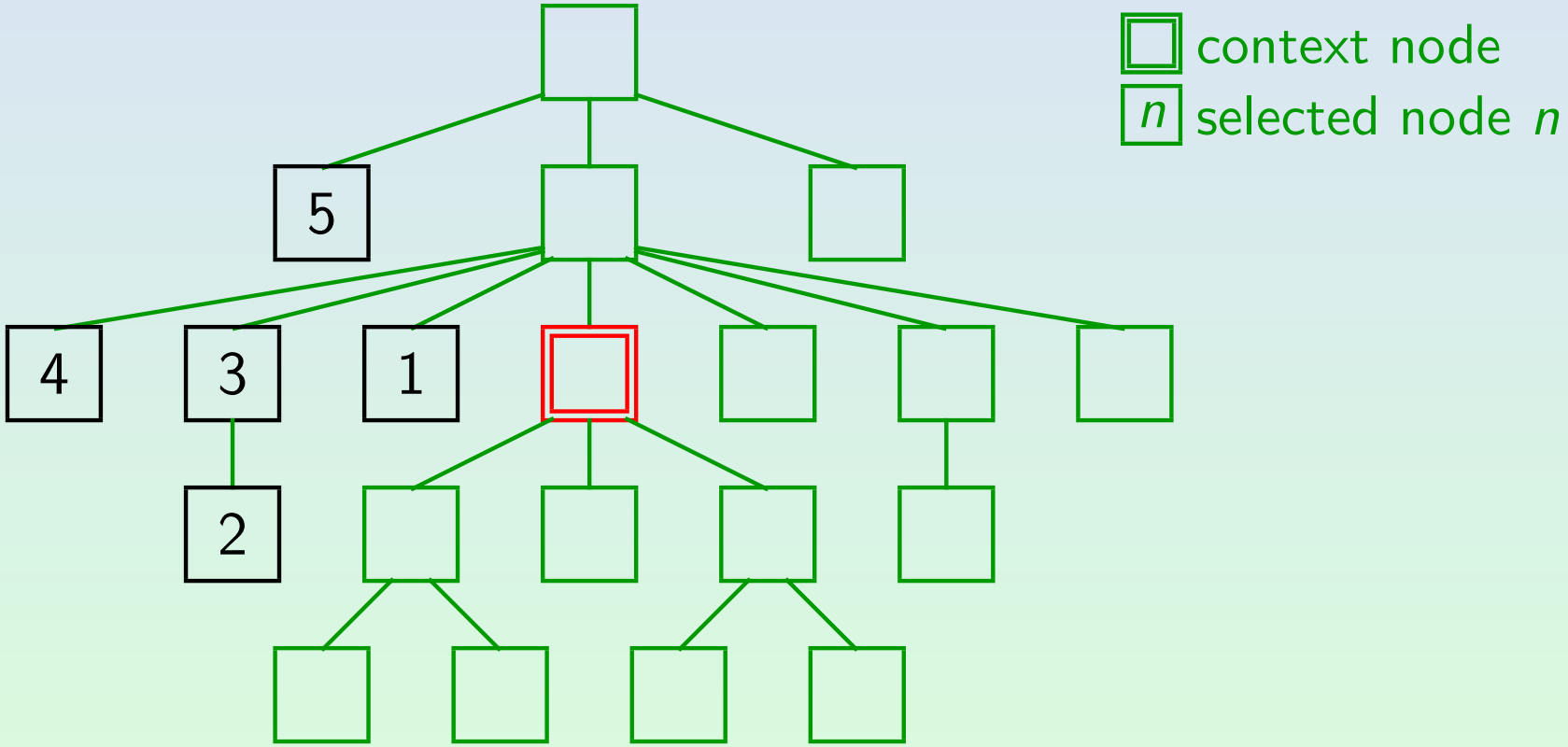
XPath Axis (14)

preceding-sibling:



XPath Axis (15)

preceding:



Contents

Node Tests (1)

- A node test is a name test or a node type test.
- In XPath 1.0, a name test had one of the forms
  - `QName` (local name or prefix:local name)

    Note that the standard default namespace declaration does not apply to XPath. Furthermore note that the namespace URIs are compared, not the prefix.
  - `NCName:*` (arbitrary name in given namespace)
  - `*` (no restriction)
- If a name test is used, the node type must be the principal type of the axis, which is "element" for all axis except the attribute and the namespace axis.

Node Tests (2)

- In XPath 1.0, the node types that could be used as node tests were:
  - `comment()`
  - `text()`
  - `processing-instruction()`
  - `processing-instruction('target')`
  - `node()`: All nodes reachable by the given axis.

  There, the node type is e.g. "`comment`", and the "`()`" makes it a node test. The problem is that there could be an element type "`comment`", and the "`()`" distinguishes the node type test from the name test.

  There were no node type tests for attribute and namespace nodes, because they are accessed via specific axis, and for document nodes, because this is accessed via "`/`".

Node Tests (3)

- In XPath 2.0, sequence type syntax was introduced.
  It defines a notation (name) for sequence types.
- Possible sequence types are:
  - `empty-sequence()`
  - A node kind test (see below), optionally followed by an occurrence indicator (`?`, `*`, or `+`)
  - `item()` with an optional occurrence indicator

    Remember that an item is a node or an atomic value.
  - an atomic type name (e.g., `xs:integer`) with an optional occurrence indicator.

Node Tests (4)

- The node kind tests in XPath 2.0 are:
  - `element(*)`: any element node

    This matches any element node. In an example, also `element()` is used, but the formal grammar does not seem to allow this.
  - `element(Name)`

    This matches any element node with the given name (QName).
  - `element(Name, Type)`, `element(*, Type)`

    This matches an element node with the given name (or any name in case of `*`) that is annotated with the type `Type`, or with a type derived from `Type`. The type can be followed by `?`, which permits nilled nodes. Otherwise nilled nodes would not match.

Node Tests (5)

- Node kind tests in XPath 2.0, continued:
  - `schema-element(Name)`

    This matches an element called `Name` or declared in a substitution group below `Name`. In addition, it must have the data type declared in the schema for the `Name`, or a more specific type. It can possibly be nilled if the element is declared as nillable. Basically, there must be a top-level declaration for `Name` in the schema, because the names of locally declared element types are implementation-dependent.
  - `attribute(*)`
  - `attribute(Name)`
  - `attribute(Name, Type)`, `attribute(*, Type)`
  - `schema-attribute(Name)`

Node Tests (6)

- Node kind tests in XPath 2.0, continued:
  - `document-node()`

    One can also use e.g. `document-node(element(GRADES-DB))`, and the same with the other forms of `element` and `schema-element` tests. The element test refers to the unique child element (document element). If there should be several child elements, the test fails.
  - `processing-instruction(Name)`

    The name can be a QName or (for backward compatibility) also a string. It is optional, i.e. `processing-instruction()` is also possible.
  - `comment()`
  - `text()`
  - `node()`

Node Tests (7)

- The node name tests in XPath 2.0 are as shown above for XPath 1.0, only the new wildcard `*:...` was added (given local name, arbitrary namespace):
  - `QName` (i.e. `NCName` or `NCName:NCName`).
  - `*`
  - `NCName:*`
  - `*:NCName` (new in XPath 2.0)

Contents

Predicates (1)

- A predicate `[...]` filters an input sequence.
- It checks a condition for each item in the input sequence and yields an output sequence that contains only those items for which this condition is true.
- For each item in the input sequence, an "inner focus" is computed, i.e. the evaluation context is changed. With this context, the expression in `[...]` is evaluated.
- Once this is finished, one returns to the original context, i.e. the "outer context" (like a stack).

Predicates (2)

Evaluation of `E1[E2]`:

- `E1` is evaluated, let the result be sequence s.
- For each item x in s, an inner focus is computed as follows: The context item is x, the context size is the length of s, and the context position is basically the position of x in s.

  More precisely: If this predicate appears in a forward axis step, the context position is the position x would have if s were sorted in document order. If the predicate appears in a reverse axis step, the context position is the position x has between the nodes in s in inverse document order. If the predicate is not in a step, the context position is the position of x in s.

Predicates (3)

Evaluation of `E1[E2]`, continued:

- For each x in `s` (result of evaluating `E1`), `E2` is evaluated in the focus described above.
  - If the result is a numeric value, it is compared with the context position in this inner focus. If they are equal, x is appended to the output sequence.
  - Otherwise, the effective boolean value of the result is computed (see next slide). If it is true, x is appended to the output sequence.

Effective Boolean Value (1)

- Effective boolean value of an expression that returns value x:
  - If x is the empty sequence, the result is false.
  - If x is a sequence, the first item of which is a node, the result is true.
  - If x is a value of type `boolean` (or derived from `boolean`), the result is x.

    Formally, x is a singleton sequence containing a boolean, but singleton sequences are identified with the item they contain. Note that sequences of two or more atomic values lead to an error (see next slide), while sequences of two or more nodes are no problem (and are treated as true, see above).
  - ... (continued on next slide)

Effective Boolean Value (2)

- Effective boolean value of x, continued:
  - If x is a `string` (or `anyURI`, `untypedAtomic` or derived from one of these), the result is false if it is the empty string, true otherwise.
  - If x belongs to a numeric type, the result is false if it is equal to 0 or `NaN`, true otherwise.

    For use in predicates, numeric values are interpreted as indexes, but the effective boolean value is not only used in predicates.
  - In all other cases, a type error is raised.
- Formally, this very generous conversion to `boolean` is done by the function `boolean(`x`)`.
- In many contexts, it is called implicitly.

Subtle Differences I

- Suppose that `STUDENT` has an attribute `GUEST` of type `boolean`. Then `[attribute::GUEST]` will be true when there is a `GUEST` attribute node, even if its value is false.

  One must explicitly take the value of the attribute with the `data(...)` function. Otherwise it checks only that the attribute node exists (which might be automatically inserted by applying a default value).

- The effective boolean value of `"false"` (a string) is true.

  `boolean("false")` is true, but `xs:boolean("false")` is false.

Contents

Abbreviated Syntax

- `attribute::` can be abbreviated to "`@`".
- If no axis is given, the default axis is
  - "`child::`", unless the node test of that step is "`attribute(...)`" or "`schema-attribute(...)`".
  - In that case, the default axis is "`attribute::`".
- "`//`" is replaced by "`/descendant-or-self::node()/`".

  However, this may only be applied to a path expression that consists of something else besides "`//`". "`//`" by itself is not a legal path expression. In contrast, "`/`" is allowed.
- The step "`..`" is short for "`parent::node()`".

Meaning of Absolute Paths

- An absolute path can be understood as a relative path with first step

  ```
  root(self::node()) treat as document-node()
  ```

- Thus, it determines the root of the tree in which the context node is.

  E.g., by following the parent-link.

- This root node must be a document node, otherwise a runtime error occurs.

Exercise (1)

```
<?xml version="1.0"?>
<BOOKLIST>
    <BOOK ISBN="0-13-014714-1" PAGES="1074">
        <AUTHOR FIRST="Paul" LAST="Prescod"/>
        <AUTHOR FIRST="Charles" LAST="Goldfarb"/>
        <TITLE>The XML Handbook - 2nd Edition</TITLE>
        <PUBL DATE="19991112">Prentice Hall</PUBL>
        <NOTE>Contains CD.</NOTE>
    </BOOK>
    <BOOK ISBN="1-56592-709-5" PAGES="107">
        <AUTHOR FIRST="Robert" LAST="Eckstein"/>
        <TITLE>XML Pocket Reference</TITLE>
        <PUBL DATE="19991001">O'Reilly</PUBL>
    </BOOK>
</BOOKLIST>
```

Exercise (2)

- What is the full version of the following expression?

  ```
  /*//AUTHOR/@LAST
  ```

- Please write an XPath expression for:
  - Print the last names of all authors.

    Assume that the context node is the document node and that it suffices to select the attribute nodes, and not necessarily take their value. E.g. `<xsl:value-of select="..." separator=","/>` would automatically take the value of the attribute nodes.

- What is the difference between the XPath expressions `//TITLE` and `//TITLE/text()`?

Subtle Differences II

- Note the difference between:
  - `//A[1]`: This selects all `A`-elements that are the first `A`-child of their parent.

    `//A[1]` stands for `/descendant-or-self::node()/child::A[1]`.

    Thus, `1` is the position for a `child`-step.
  - `/descendant::A[1]`: This selects only the first `A`-element in the entire document.
- Note also that these are not the same:
  - `//A[1]`: (as above, possibly many elements).
  - `(//A)[1]`: Only first `A`-element in document.

    Here, `[1]` applies to the entire sequence returned by `//A`.

References

- Anders Berglund, Scott Boag, Don Chamberlin, Mary F. Fernández, Michael Kay, Jonathan Robie, Jérôme Siméon (Editors): XML Path Language (XPath) 2.0. W3C Recommendation, 23 January 2007. [http://www.w3.org/TR/xpath20/]
- Mary Fernández, Ashok Malhotra, Jonathan Marsh, Marton Nagy, Norman Walsh (Ed.): XQuery 1.0 and XPath 2.0 Data Model (XDM). W3C Recommendation, 23 Jan. 2007, [http://www.w3.org/TR/xpath-datamodel/]
- Ashok Malhotra, Jim Melton, Norman Walsh (Ed.): XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Recommendation, 23 January 2007. [http://www.w3.org/TR/xpath-functions/]
- G. Ken Holman: Definitive XSLT and XPath. Prentice Hall, 2002, ISBN 0-13-065196-6, 373 pages.
- Michael Kay: XPath 2.0 Programmer's Reference. Wiley/Wrox, 2004, ISBN 0-7645-6910-4, 552 pages.
- Michael Kay: XSLT 2.0 Programmer's Reference, 3rd Edition. Wiley/Wrox, 2004, ISBN 0-7645-6909-0, 911 pages.
- Miloslav Nic, Jiri Jirat: XPath Tutorial. Zvon [http://www.zvon.org/xxl/XPathTutorial/General/examples.html]