

# Stretching XPath

Three Testing Tales

Amanda Galtman

July 2024

# Three Testing Tales

**Things motivated by one purpose can be useful for another purpose.**

1) The Cupboard Was Bare

*Emptiness shouldn't be a surprise*

2) Good Functions Make Good Neighbors

*Tests shouldn't interfere with each other*

3) Schema and Variations

*Simplify maintenance of test artifacts*

# Tale 1

The Cupboard Was Bare

# Tale 1: The Cupboard Was Bare

- Empty sequences are normal, but...
- *Unexpectedly* empty sequences can cause bad outcomes
  - Test passes without serving its purpose
  - Test fails confusingly
- **Objective:** Create sequences more defensively

# Tale 1 Background

- Setting: XSpec for XSLT or XQuery
- XPath expression → Sequence
  - Parameter value
  - Expected result
- The sequence is empty —Surprise!

# Problem to Solve

- Detection
  - Mistake → ( ) → bad test outcomes
- Want unobtrusive solution

# How Can a Surprise Happen?

Maybe path `a/b/c[3]/d` should have said

- `a/b/c[2]/d`
- `a/b/x[3]/d`
- `a/b/c[3]/y/d`

Or, maybe the XML document is wrong.

# Bad Test Outcome (a)

- Objective: Verify that something does not exist
  - Sequence of paras in 3rd section is ( )
- Something upstream doesn't exist, due to a mistake
  - There *is* no 3rd section
- ( ) equals ( ) → Test passes 😞
  - Didn't serve its purpose, no feedback for us



# Bad Test Outcome (b)

- Objective: Pass parameter to function and verify output
- Mistake in parameter construction → Empty
- Function output is unexpected
- Test fails, alerting us to a problem 😊
- But *why* did it fail? 😞

# Prevention via XSpec syntax

- Extra `<x:expect>` to check if something exists
  - Clutter, repeated path
- Declare data type that can't be empty, e.g., `as="node()"`
  - For partial-path sequence, need variable → Clutter?

# Prevention via XPath

- Want an unobtrusive solution without extra elements
- XPath functions → *arg* or error
  - **exactly-one**(*arg*)
  - **one-or-more**(*arg*)
- XPath expression → *expr1* or error
  - *expr1* **treat as** *type1*

# Why Might this Solution Be Surprising?

Typical use case is about static type checking

- "Trust me, this will be non-empty at run time"
- Processor suppresses a static error
- If you were wrong, processor raises run-time error

# How the Solution Looks in XSpec (fcn)

Wrap the function call around a path:

```
<x:expect label="The third section has no paragraphs"  
test="empty(exactly-one($x:result/topic/section[3])/para)"/>
```

....

```
<x:param name="items" href="test-document.xml"  
select="one-or-more(topic/section/table)"/>
```

# How the Solution Looks in XSpec (expr)

Insert **treat as**, adding parentheses if needed:

```
<x:expect label="The third section has no paragraphs"
test="empty(($x:result/topic/section[3] treat as
element(section))/para)"/>
```

....

```
<x:param name="items" href="test-document.xml"
select="topic/section/table treat as element(table)+"/>
```

# How the Solution Behaves

- If condition is met: Pass-through
- Otherwise: Test halts with clear error message
  - (a) Tells us something is wrong
  - (b) Hints that the problem is in the test, not the XSLT/XQuery
- We still have to find *which* sequence caused the error

# Broader Lesson Beyond Testing

- "What if this sequence *[or partial path]* is empty?"
- If the answer is "Oh no!", code defensively
  - Insert call to `one-or-more` or `exactly-one`
  - Use `treat as` expression
  - Use `as` attribute (*on some element*) to declare a data type that excludes ( )



# Tale 2

Good Functions Make Good Neighbors



# Tale 2: Good Functions Make Good Neighbors

- Combining XML fragments in one document is compact, but...
- They might not coexist well during testing
- **Objective:** Compact *and* harmonious

# Tale 2 Background

- Setting: XSpec for XSLT
- Multiple tiny test cases
  - Apply templates to tiny XML fragments
  - Tiny result
  - Similarity across test cases
- e.g, Attr → Attr, <a> → <b> with markup variations

# Problems to Solve

- *Initial problem:* Testing cases individually is too much code
- Solution: Combine the test cases
  - One XML document, one XSpec scenario
- *New problem:* Test cases could interfere with each other

# What Could Possibly Go Wrong?

If XSLT code accesses arbitrary parts of tree,...

- Markup from one test case could influence another test case
- Combined document might violate assumptions
  - e.g., multiples of something assumed unique
  - Could have unexpected results or errors.

*Real concern? Terrible idea?*

# Example

- Link to titled note produces generated text
- Want to test gentext with markup variations in title
- Combined test document is a note with multiple titles
  - Siblings
  - (Not schema-valid)
- Point XSpec at titles
  - Expect to get sequence of mini-documents for gentext

# Example XSpec Code (sibling titles)

```
<x:scenario label="Test multiple note titles">  
  <x:context select="//d:title" mode="xref-to">  
    <d:note>  
      <d:title>Simple Title</d:title>  
      <d:title>Title with <d:code>code</d:code></d:title>  
    </d:note>  
  </x:context>  
  
</x:scenario>
```

# Example XSpec Code (sibling titles)

```
<x:scenario label="Test multiple note titles">
  <x:context select="//d:title" mode="xref-to">
    <d:note>
      <d:title>Simple Title</d:title>
      <d:title>Title with <d:code>code</d:code></d:title>
    </d:note>
  </x:context>
  <x:expect label="Show me the input titles as strings"
    test="$x:context ! string(.)" as="xs:string*" />
  <x:expect label="Show me the actual result" />
</x:scenario>
```



# Example Results

- Context has both titles 😊
- Actual result is the 1st title multiple times 😞
- Why?
  - Title from 1st test case interfered with 2nd test case
  - XSLT assumptions violated

**Show me the titles as strings**

**Result**

```
'Simple Title',  
'Title with code'
```

**Show me the actual result**

**Result**

XPath `/node()` from:

```
Simple TitleSimple Title
```

# Fix Problem in XSpec? In XML?

- No XSpec syntax for deleting unwanted `<title>` or constructing new `<note>`
- Can use helper functionality (`<x:helper>`)
  - Not exactly lightweight for this example
- Can enlarge XML document: multiple `<note>` elements
  - Sure, but let's look harder to solve with existing document

# Fix via XPath/XSLT

- **snapshot**(*arg*) copies subtree, preserving ancestry
- **copy-of**(*arg*) copies subtree alone
- Siblings omitted
  - Sibling titles caused our interference
  - Tiny test cases might be siblings in combined document

```
<d:note>  
<d:title>1</d:title>  
<d:title>2</d:title>  
</d:note>
```



```
<d:note>  
<d:title>2</d:title>  
</d:note>
```

# Why Might this Solution Be Surprising?

Typical use case is about streaming (XSLT 3.0)

- Streaming restricts access to nodes of tree
- Buffering copy of subtree enables freer access to it
  - Memory considerations in streaming
  - Here, **isolation** is key
- Sometimes you need ancestry, sometimes not

# How the Solution Looks in XSpec

```
<x:context select="//d:title/snapshot()" mode="xref-to">
  <d:note>
    <d:title>Simple Title</d:title>
    <d:title>Title with <d:code>code</d:code></d:title>
  </d:note>
</x:context>
```

# How the Solution Behaves

- Actual result has correct gentext mini-documents 😊
- (Verifying sequence of document nodes scalably is a bit tricky)

**Show me the actual result**

**Result**

XPath `/node()` from:

```
Simple TitleTitle with <code class="code">code</code>
```

# How the Solution Behaves (cont'd.)

- `<result>` is an ad hoc stand-in for document node in expected result.
- Arrays keep separate node collections apart. (`Simple TitleTitlewith` fails.)
  - ( `["Simple Title"]`, `["Title with ", <code.../>]` )

```
<x:expect label="Verify all titles"
  test="for $r in $x:result return [$r/node()]"
  select="for $r in /result return [$r/node()]">
  <result>Simple Title</result>
  <result>Title with <code class="code">code</code></result>
</x:expect>
```

# Broader Lesson Beyond Testing

- When nodes become neighbors for convenience, consider if further processing requires isolating them
- `snapshot` and `copy-of` make good neighbors
  - Not an everyday use case, but when it's a good fit, you'll like how easy it is

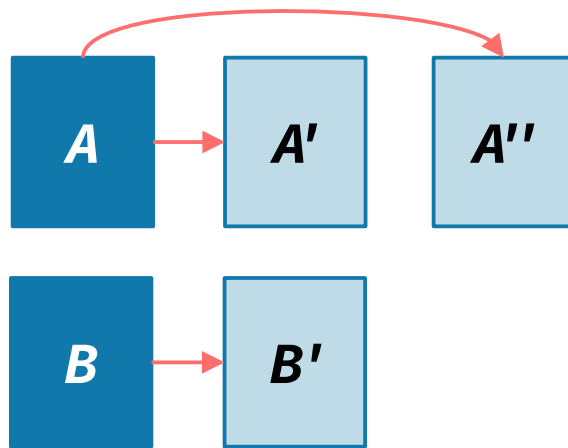


# Tale 3

## Schema and Variations

# Tale 3: Schema and Variations

- Schema testing requires validating documents, but...
- How do you maintain them all?
- **Objective:** Keep some as sources, derive others



# Tale 3 Background

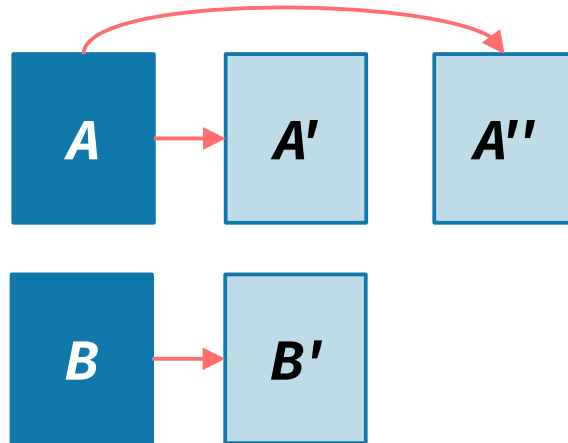
- Setting: BaseX or XSpec for testing a schema
- Large set of documents to validate
- Invalid documents are invalid for 1 reason
  - Nearly identical to a valid document in the set

# Problem to Solve

- Maintaining documents
- Maintaining tiny diffs between  $A$  and  $A'$ ,  $A$  and  $A''$ , etc.

# Main Solution Idea

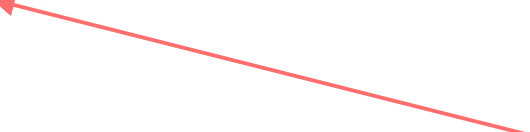
- Derive  $A'$  and  $A''$  from  $A$
- Derivation code must be easy to write/maintain



# Example in BaseX

- Helper functions that validate document and verify result
- For each document in set, call helper from small test function

```
declare %unit:test function mytest:valid-sample-doc() {  
  let $s := doc('original/valid-123.xml')  
  (: As is, file is valid :)  
  return  
    mytest:expect-valid($s)  
};
```



Want to derive variation document and call function on that instead.

# Ad Hoc Solution Option

- (Separate process from test running, or not?)
- Identity plus specify/implement desired variation?

# XQuery Update Standard

- Complements XQuery standard
- Insert, delete, rename, and replace nodes
- BaseX, Saxon-EE, and maybe others
- Aside: Similar operations in `<saxon:update>` XSLT extension, GitHub libraries



# Why Might this Solution Be Surprising?

Typical official use cases are about *updating* something:

- Update database or other persistent storage
- Add new information
- Refresh status

# Our Use Case Is Slightly Different

- Document  $A'$  is not fresher than  $A$
- Retain both, for validation testing
- Derive valid document from invalid one, or vice versa

# How the Solution Behaves

- 1) Read document from file
  - 2) Modify in memory using XQuery Update expressions
    - Simple and few
    - Invalid for 1 reason
  - 3) Write to a different file
- 
- 4) Validate from file
  - 5) Verify validation results
- OR
- 3) Validate in memory
  - 4) Verify validation results

# How a Solution Looks in BaseX (same process)

```
declare %unit:test function mytest:xqupdate-makes-invalid-literal()  
{  
    ... $s := doc('original/valid-123.xml')  
  
    Derivation goes here  
  
    return  
        mytest:expect-invalid($s)  
};
```

# How a Solution Looks in BaseX

```
declare %unit:test function mytest:xqupdate-makes-invalid-literal()  
{  
  copy $s := doc('original/valid-123.xml')  
  modify (  
    (: Renaming <code> as <literal> makes the file invalid :)  
    rename node $s/d:article//d:code  
      as 'd:literal'  
  )  
  return  
    mytest:expect-invalid($s)  
};
```

# Broader Lessons Beyond Testing

- Applying *and maintaining* small tweaks to many documents?
- Consider a standard solution
  - Or a partial imitation, if that's a better fit

# The Moral of the Story

- Is non-emptiness important? Say it in code!
  - `one-or-more` and `exactly-one`
  - `treat as`
- Free a subtree from interfering siblings/ancestors
  - `snapshot` and `copy-of`
- Look beyond suggestive terminology
  - You decide if a "difference" counts as an "update"

# Thank You

## Questions?



# Saxon-EE XSLT Update Extension

```
<xsl:result-document href="gen/invalid-123-literal.xml">
  <saxon:update select="doc('original/valid-123.xml')">
    <saxon:rename select="/d:article//d:code" to="'d:literal'"/>
  </saxon:update>
</xsl:result-document>
<xsl:result-document href="gen/valid-456-abstract-moved.xml">
  <saxon:update select="doc('original/invalid-456.xml')">
    <saxon:insert select="/d:article/d:info/d:author" position="before">
      <xsl:sequence select="/d:article/d:abstract"/>
    </saxon:insert>
    <saxon:delete select="/d:article/d:abstract"/>
  </saxon:update>
</xsl:result-document>
```

# Saxon-EE XSLT Update Extension in XSpec for Schematron

```
<!-- XSpec test file -->
<x:helper stylesheet="saxon-update-helper.xsl"/>
<x:scenario label="No article title">
  <x:context href="valid-123.xml" select="dd:rm-article-title(/)" />
  <x:expect-assert id="a001"/>
</x:scenario>
```

```
<!-- XSLT helper file -->
<xsl:function name="dd:rm-article-title" as="document-node()">
  <xsl:param name="doc" as="document-node()" />
  <saxon:update select="$doc">
    <saxon:delete select="/article/title"/>
  </saxon:update>
</xsl:function>
```