Saxon - Bug #5110

XPDY0002 The context item is absent, so position() is undefined for use of position() inside predicate

2021-09-30 19:21 - Martin Honnen

Status:	Resolved	**Start date:**	2021-09-30
Priority:	Normal	**Due date:**	
Assignee:	Michael Kay	**% Done:**	0%
Category:	XQuery conformance	**Estimated time:**	0:00 hour
Sprint/Milestone:	10.6	**Spent time:**	0:00 hour
Legacy ID:		**Fixed in Maintenance Release:**	
Applies to branch:	10, 11, trunk	**Platforms:**	
Fix Committed on Branch:	10, trunk		

Description

I get the error

```
Error on line 8 column 29 of powerset1.xq:
  XPDY0002  The context item is absent, so position() is undefined
     Focus: absent
     Local variables
        $counter = 0
        $input =  ("A", "B", "C", ... [4])
        $counter-bin = xs:base64Binary("AA==")
     invoked by unknown caller (class net.sf.saxon.value.MemoClosure)
     Focus: absent
     Local variables
        $counter = 0
        $input =  ("A", "B", "C", ... [4])
     invoked by function call at file:/C:/Users/Martin%20Honnen/OneDrive/Documents/XQuery/powerset
/./powerset1.xq#17
     Focus: absent
     Local variables
        $a = []
        $i = 0
        $input =  ("A", "B", "C", ... [4])
     invoked by unknown caller (class net.sf.saxon.functions.hof.UserFunctionReference$BoundUserFu
nction)
```

running Saxon 10.6 EE Java against the XQuery

```
declare namespace math = 'http://www.w3.org/2005/xpath-functions/math';
declare namespace bin = 'http://expath.org/ns/binary';
declare namespace array = 'http://www.w3.org/2005/xpath-functions/array';

declare function local:combination($counter as xs:integer, $input as item()*) as item()* {
   let $counter-bin := bin:from-octets($counter)
   return $input[
     let $j:= position() - 1,
     $bin := bin:and($counter-bin, bin:shift(bin:hex('1'), $j))
     return bin:to-octets($bin) > 0]
 };

declare function local:powerset($input as item()*) as array(item()*) {
  fold-left(
    0 to (xs:integer(math:pow(2, count($input))) - 1),
    [],
    function($a, $i) { array:append($a, local:combination($i, $input)) }
   )
```

```
 };

local:powerset(('A', 'B', 'C', 'D'))
```

The code runs fine with BaseX, outputting [(), "A", "B", ("A", "B"), "C", ("A", "C"), ("B", "C"), ("A", "B", "C"), "D", ("A", "D"), ("B", "D"), ("A", "B", "D"), ("C", "D"), ("A", "C", "D"), ("B", "C", "D"), ("A", "B", "C", "D")].

History

#1 - 2021-10-01 09:10 - Martin Honnen

If I turn off any optimization with e.g. -opt:0 (and add declare namespace output = "http://www.w3.org/2010/xslt-xquery-serialization"; declare option output:method 'adaptive';) to code gives the wanted result.

#2 - 2021-10-06 10:29 - Michael Kay

Added as a test case to EXPath-binary-extra/EXPath-binary-powerset.

Problem reproduced.

#3 - 2021-10-06 11:13 - Michael Kay

In FilterExpression.java, evaluating $input[....], it has wrongly decided that the value of the predicate is independent of the focus.

When first calculated in FilterExpression.typeCheck(), the value of filterIsIndependent' is correctly set to false(andfilterIsPositionalis correctly set totrue`).

But in the optimize() phase, the re-calculation of filterIsPositional at line 389 wrongly sets it to false. Though on exit from optimize(), filterIsIndependent' is still correctly set to false`.

But then we attempt to rewrite the FLWOR expression as a "for" or "let" expression, which re-does the typeCheck() on the FilterExpression, and this time filterIsIndependent becomes true.

#4 - 2021-10-06 11:20 - Michael Kay

Note that the test runs correctly on the 11 branch.

I suspect (with no evidence) that the difference might be that the calls on bin:xx() functions are now implemented as integrated extension functions rather than reflexive extension functions, which may mean that they are handling context-dependent subexpressions better.

#5 - 2021-10-06 11:26 - Michael Kay

Correction: it doesn't work on the 11 branch. It no longer crashes, but it produces wrong output. The result on SaxonJ is

```
AA BB AA BB CC AA CC BB CC AA BB CC DD AA DD BB DD AA BB DD CC DD AA CC DD BB CC DD AA BB CC DD
```

and on SaxonCS we get

```
A B A B C A C B C A B C D A D B D A B D C D A C D B C D A B C D
```

(which is the same sequence without the doubling)

#6 - 2021-10-06 12:09 - Michael Kay

If I run under SaxonCS with !method=adaptive, the output is

```
[(),"A","B",("A","B"),"C",("A","C"),("B","C"),("A","B","C"),"D",("A","D"),("B","D"),("A","B","D"),("C","D"),("A","C","D"),("B","C","D"),("A","B","C","D")]
```

which is the same as BaseX. Phew!!!

#7 - 2021-10-06 15:30 - Michael Kay

The difference between the J and CS behaviour on the 11 branch is purely by chance. There is a bug in UTF8Writer.write(UnicodeString) whereby if the UnicodeString is a UnicodeChar, then the character is written twice. This bug doesn't affect the query as I was running it in SaxonCS because the output is being sent to a writer, not a stream, so the UTF8Writer is not used. But the UTF8Writer is present in SaxonCS and the bug could manifest itself under different circumstances.

#8 - 2021-10-20 13:40 - Michael Kay

Picking this one up after a gap: current status is that it works correctly on 11.x, but fails on 10.x.

The -explain output for the local:combination() function is

```
<declareFunction name="local:combination" arity="2" tailRecursive="false">
```

```
    <let baseUri="file:/Users/mike/bugs/2021/5110-Honnen/test.xq"
         ns="array=~ bin=http://expath.org/ns/binary err=~ fn=~ local=http://www.w3.org/2005/xquery-local-func
tions math=~ saxon=~ xs=~ xsi=~ xml=~"
         line="6"
         var="Q{}counter-bin"
         slot="2"
         eval="8">
      <javaCall name="Q{http://expath.org/ns/binary}from-octets" arg0type="1ADI">
        <varRef name="Q{}counter" slot="0"/>
      </javaCall>
      <filter line="7" flags="ib">
        <varRef name="Q{}input" slot="1"/>
        <gc line="10" op="&gt;" card="N:1" comp="CAVC">
          <javaCall name="Q{http://expath.org/ns/binary}to-octets" arg0type="1A2">
            <check card="1" diag="0|0||bin:to-octets">
              <javaCall line="9"
                        name="Q{http://expath.org/ns/binary}and"
                        arg0type="?A2"
                        arg1type="?A2">
                <varRef name="Q{}counter-bin" slot="2"/>
                <javaCall name="Q{http://expath.org/ns/binary}shift"
                          arg0type="?A2"
                          arg1type="1ADI">
                  <javaCall name="Q{http://expath.org/ns/binary}hex" arg0type="1AS">
                    <str val="1"/>
                  </javaCall>
                  <arith line="8" op="-" calc="i-i">
                    <fn name="position"/>
                    <int val="1"/>
                  </arith>
                </javaCall>
              </javaCall>
            </check>
          </javaCall>
          <int val="0"/>
        </gc>
      </filter>
    </let>
  </declareFunction>
```

which looks perfectly OK. And yet the query works if I set -opt:0. The only significant optimisations seem to be (a) the conversion of a FLWOR expression to a let expression, and (b) the inlining of variables $bin and $j - which looks harmless.

#9 - 2021-10-20 13:45 - Michael Kay

Actually, the thing that's critically wrong in the -explain output is the flags="i" on the filter expression. This means filterIsIndependent - the optimizer has concluded that the value of the predicate does not depend on the focus and can therefore be evaluated in the context of the outer expression - which fails because the outer context has no context item or position. So it's the calculation of dependencies that's wrong -- as explained in comment #3.

#10 - 2021-10-20 18:43 - Michael Kay

Getting closer, but it's tough going.

In the FLWORExpression

```
     let $j:= position() - 1,
         $bin := bin:and($counter-bin, bin:shift(bin:hex('1'), $j))
     return bin:to-octets($bin) > 0]
```

both the variables get inlined in turn, producing

```
bin:to-octets(bin:and($counter-bin, bin:shift(bin:hex('1'), position() - 1)) ) > 0]
```

and for some reason the cached properties of the GeneralComparison bin:to-octets($bin) > 0] (with no focus dependencies) are retained through the rewrite, so the final expression is considered to have no focus dependency either.

Note that because there are two variables, we are taking a slightly different route from normal - with one variable, we would probably turn the FLWORExpression into a LetExpression before eliminating the variable. This is no doubt why the problem has never been encountered before.

#11 - 2021-10-20 18:47 - Michael Kay

Fixed with two changes:

(a) in ExpressionTool.replaceSelectedSubexpressions(), change the recursive call from

`replaced = replaceSelectedSubexpressions(....)`

to

`replaced |= replaceSelectedSubexpressions(...)`

so that the variable replaced is true if any operand has been replaced, rather than just the last one.

(b) In ExpressionTool.replaceVariableReferences(), if the call on replaceSelectedSubexpressions() returns true, reset cached properties of the expression. For safety we do this throughout the subtree.

Note that although the test case didn't fail in 11.x, the code here is exactly the same and the same changes should be applied.

There's a secondary issue which is that these optimization rewrites aren't being traced when -explain is used.

#12 - 2021-10-20 19:07 - Michael Kay

- Status changed from New to Resolved

- Applies to branch 11, trunk added

- Fix Committed on Branch 10, trunk added

Fix committed on 10.x and 11.x branches.

#13 - 2021-11-11 19:31 - Michael Kay

- Assignee set to Michael Kay