How to Create XSLT Functions that Manipulate Functions (a.k.a. Higher-Order Functions)

Roger L. Costello
September 2010

Introduction

A powerful programming technique is to create functions that can accept functions as arguments or return functions as values. Functions that manipulate functions are called *higher-order* functions. This article shows how to implement higher-order functions using XSLT 2.0.

I learned this technique from Dimitre Novatchev. Thanks Dimitre!

Motivation for Higher-Order Functions

Example: Suppose you want to square each item in this sequence:

```
(2, -2, 4, -16, 22, -890)
```

The result of applying a square function to each item is this new sequence:

```
(4, 4, 16, 256, 484, 792100)
```

Example: Next, suppose you want to take the absolute value of each item in this sequence:

```
(15, -22, -74, -106, 232, -89)
```

The result of applying the absolute value function to each item is this new sequence:

```
(15, 22, 74, 106, 232, 89)
```

Do you see a pattern emerging from these two examples?

The pattern is this: Apply a function to a sequence.

Thus, a useful programming abstraction is a function that can take any function, `proc`, and any sequence, `sequence`, and apply `proc` to `sequence`.

This function is commonly called a "map" function. The `map` function takes two arguments:

1. A function (`proc`)
2. A sequence of values (`sequence`)

The `map` function applies (maps) `proc` to each value in `sequence`.

Example: `map(square, (2, -2, 4, -16, 22, -890))`

`returns`

```
(4, 4, 16, 256, 484, 792100)
```

Example: `map(abs, (15, -22, -74, -106, 232, -89))`

`returns`

```
(15, 22, 74, 106, 232, 89)
```

Implementing Higher-Order Functions in XSLT

Let's first implement `square` and `abs`:

Step 1: Create an XSLT file (sequence-functions.xsl) and declare a namespace:

```
xmlns:f="http://www.data-structures.org/sequence/"
```

Step 2: For each function create a variable with a value that is an element (whose name is the name of the function):

```
<xsl:variable name="square" as="element()"><f:square /></xsl:variable>

<xsl:variable name="abs" as="element()"><f:abs /></xsl:variable>
```

Be sure to associate each element with the namespace declared in Step 1.

The value of $square is <f:square />. The value of $abs is <f:abs />.

Step 3: Create a template rule that matches on the variable's value:

```
<xsl:template match="f:square">

    ...

</xsl:template>


<xsl:template match="f:abs">

    ...

</xsl:template>
```

Step 4: Each template rule will be invoked with a sequence of integers and the result will be a sequence of integers:

```
<xsl:template match="f:square" as="xs:integer*">
    <xsl:param name="sequence" as="xs:integer*"/>

    ...

</xsl:template>


<xsl:template match="f:abs" as="xs:integer*">
    <xsl:param name="sequence" as="xs:integer*"/>

    ...

</xsl:template>
```

Step 5: Each template rule simply invokes a function with the same name, passing it the sequence:

```
<xsl:template match="f:square" as="xs:integer*">
    <xsl:param name="sequence" as="xs:integer*"/>

    <xsl:sequence select="f:square($sequence)" />

</xsl:template>


<xsl:template match="f:abs" as="xs:integer*">
    <xsl:param name="sequence" as="xs:integer*"/>
```

```
        <xsl:sequence select="f:abs($sequence)" />

    </xsl:template>
```

Step 6: Implement the `square` and `abs` functions:

```
    <xsl:function name="f:square" as="xs:integer*">
        <xsl:param name="sequence" as="xs:integer*"/>

        <xsl:for-each select="$sequence">
            <xsl:sequence select=". * ." />
        </xsl:for-each>

    </xsl:function>


    <xsl:function name="f:abs" as="xs:integer*">
        <xsl:param name="sequence" as="xs:integer*"/>

        <xsl:for-each select="$sequence">
            <xsl:sequence select="
                    if (. lt 0) then xs:integer(- .) else ." />
        </xsl:for-each>

    </xsl:function>
```

Recap: We want to create the `square` and `abs` functions. To enable them to be used as arguments to higher order functions we created 3 things:

1. A variable that has an element value, and the element is associated with a namespace.
2. A template rule. It has a parameter that is a sequence of values. The template rule simply invokes its corresponding function (e.g., the template rule for `square` invokes the `square` function)
3. The function.

Now we can create the `map` higher-order function.

Step 7: Create a second XSLT file (higher-order-functions.xsl) and declare a namespace:

```
xmlns:higher-order-function="http://www.data-structures.org/higher-order/"
```

Step 8: The `map` function is invoked with two arguments:

1. A function (`proc`)

2. A sequence of values (`sequence`)

It returns a sequence of values. Here's its signature:

```
<xsl:function name="higher-order-function:map" as="item()*">
    <xsl:param name="proc" as="element()"/>
    <xsl:param name="sequence" as="item()*"/>

    ...

</xsl:function>
```

Step 9: *This is the key thing to know about creating higher-order functions in XSLT*: The value of `proc` is one of the variables declared in Step 2. The `map` function simply uses `<xsl:apply-templates select="$proc">` to fire the template rule for the function. For example, if proc's value is $square then

```
<xsl:apply-templates select="$proc">
```

will result in firing this template rule:

```
<xsl:template match="f:square" as="xs:integer*">
```

If proc's value is $abs then

```
<xsl:apply-templates select="$proc">
```

will result in firing this template rule:

```
<xsl:template match="f:abs" as="xs:integer*">
```

The body of `map` is simply `apply-templates` with `sequence` as its argument:

```
<xsl:function name="higher-order-function:map" as="item()*">
    <xsl:param name="proc" as="element()"/>
    <xsl:param name="sequence" as="item()*"/>

    <xsl:apply-templates select="$proc">
        <xsl:with-param name="sequence" select="$sequence" />
    </xsl:apply-templates>

</xsl:function>
```

Recap: Invoke the `map` function with a variable and a `sequence`. The variable corresponds to the function you want `map` to invoke. The `map` function simply fires the template rule that matches the variable and passes `sequence` to that rule.

Lastly, let's use the map function.

Step 10: Create a third XSLT file (test.xsl) and declare the sequence and higher-order-function namespaces:

```
xmlns:f="http://www.data-structures.org/sequence/"

xmlns:higher-order-function="http://www.data-structures.org/higher-order/"
```

Step 11: Include the two XSLT files created above:

```
<xsl:include href="higher-order-functions.xsl"/>
<xsl:include href="sequence-functions.xsl"/>
```

Step 12: Invoke `map` with the `square` variable (i.e., function) and a sequence:

```
higher-order-function:map($square, (2, -2, 4, -16, 22, -890))
```

Here's the result:

4 4 16 256 484 792100

Step 13: Invoke `map` with the `abs` variable (i.e., function) and a sequence:

```
higher-order-function:map($abs, (15, -22, -74, -106, 232, -89))
```

Here's the result:

15 22 74 106 232 89

Try It Out!

Here is a zip file that contains the examples described above:

http://www.xfront.com/higher-order-functions-in-XSLT/higher-order-functions.zip