Design and Implementation of a CSS-based CODEML Viewer

(Guided Research Final Report)

Author: Rajeshwor Dhital
r.dhital@iu-bremen.de
International University Bremen
April 25, 2005

Supervisor: Prof. Dr. Michael Kohlhase

Abstract

CODEML is an XML application for describing program code while capturing both its structure and content. It was introduced by Prof. Dr. Michael Kohlhase and is loosely modeled after the MATHML format. The goal of CODEML is to enable program code to be served, received, and processed on the World Wide Web, in a similar manner as HTML does for the text. CODEML is a semantic markup and if it is to be presented to users it has to be transformed into presentational markup. So in this project we will be finding a good way to visualize CODEML. There are a variety of choices for the syntax of this presentation layer. Here, we will try to use the Cascading Style Sheet (CSS2) to achieve our goal. However as we proceed we will recognize that because of the lack of proper support of CSS by today's browsers, presentation of CODEML by CSS is very limited. We will also discuss how the structure of CODEML is quite complicated for the CSS2. There after we will explore an alternative solution based on XSL with CSS, which is still client side but works very well with CODEML and has greater flexibility and richness.

Contents:

1. Introduction to CODEML

With the popularity of the internet the kind of task performed in the web has also increased. These days it has become quite popular to share source code of a program or scripts on the web. It is very much preferred that the code is pretty printed such that the users can easily read. Until now this is only possible by web markup formats like HTML which not only has a very limited functionality but also doesn't capture the semantic of the source elements. Along with the usual text formatting tags HTML provides <CODE> tag element which is specially designed to present source codes. But this only allows you to visualize the whole source code as one unit and doesn't give the flexibility to change the presentation of the elements in the source code in different ways. To overcome this code markup language called CODEML was invented. CODEML is an XML application and is used to describe program code, capturing both its structure and content. CODEML elements are specified in such a way that it is independent of the programming language to be defined. So to present the source code using CODEML one first needs to mark it up with CODEML tags. Due to its complexity it is best that this is automatically done by a parser. So a project that tries to convert a Java source code into CODEML format has already been started. Please refer to the CODEML specification of Prof. Dr Michael Kohlhase for more detailed understanding of CODEML. Listing 1.2 shows the CODEML representation of a java code snippet presented in Listing 1.1.

Listing 1.1: A java code snippet [Koh01]

```
public int find (int x) {
    if (s [x] <0) return x;
    return find (s [x ]);
    }
```

Listing 1.2: The CODEML representation for the code snippet in Listing 1 [Koh01]

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE code SYSTEM “../../dtd/codeml.dtd">
<cpg>
      <cpg>
            <cpo>public</cpo>
            <cptype>int</cptype>
            <cpo>find</cpo>
            <cpg open="(" close=")">
                  <cptype>int</cptype>
                  <cpi>x</cpi>
            </cpg>
      </cpg>
      <cpg open="{" close="}" breakO="hard">
            <cpo>if</cpo>
            <cpg open="(" close=")">
                  <cpo>s</cpo>
                  <cpg open="[" close="]">
                        <cpi>x</cpi>
                  </cpg>
                  <cpo>&lt;</cpo>
                  <cpb type="number">0</cpb>
            </cpg>
            <cpg close=";" Cbreak="hard">
                  <cpo>return</cpo>
                  <cpi>x</cpi>
            </cpg>
            <cpg close=";" Cbreak="hard">
                  <cpo>return</cpo>
                  <cpg>
                        <cpo>find</cpo>
                        <cpg open="(" close=")">
                              <cpo>s</cpo>
                              <cpg open="[" close="]">
                                    <cpi>x</cpi>
                              </cpg>
                        </cpg>
                  </cpg>
            </cpg>
      </cpg>
</cpg>
```

2. Visualizing CODEML

With the CODEML markup format we manage to capture the semantic and structure of the source code. However, remember that the major purpose of CODEML is to provide a way to present codes on the web. Until and unless these codes can be presented in a display with nice formatting and pretty printing the source in CODEML format is of not much use. This is where this project comes into play. We will come with a way to format and display the source code (already converted to CODEML format) with pretty printing. For this to occur, there must be a step where formatting information is applied to the XML document and the semantic markup is transformed into presentational markup. There are a variety of choices for the syntax of this presentation layer. However, the following two are particularly noteworthy:

- Cascading Style Sheets (CSS)
- XSL Formatting Objects (XSL-FO)

In our case we choose CSS as the major goal of the project is to visualize CODEML using CSS. CSS is a total client side solution and is fairly supported in the latest versions of browsers like FireFox, Opera and Internet Explorer. One of the advantages of XSLT over CSS is its ability to change the markup element. However in our case we can move this task to the parser converting the source code to CODEML. This gives us one more reason to select CSS.

3. Short Introduction to CSS

CSS is a non-XML syntax for describing the appearance of particular elements in a document. It is a very straight-forward language that is easy to learn and remember. CSS was originally introduced in 1996 as a standard means of adding information to HTML documents about style properties such as fonts and borders. However, today it has been found to work better with XML than HTML itself [HAR01]. When CSS format is applied no transformation is performed. The parsed character data of the document is presented more or less exactly as it appears in the XML document. There are three levels of CSS. The first level CSS1 was quite limited and was not so complete. So a second level CSS2 was introduced. This level is the one that most browsers today support. Although the third level CSS3 is extremely powerful and rich we will focus in CSS2 as CSS3 is not yet that well supported by browsers.

A CSS style sheet contains a list of rules. Each rule gives the names of the elements it applies to and the styles to apply to those elements. One important thing to note is only one rule per element is applied even though a single element can have multiple rules. When an element has multiple CSS rule the most specific rule is chosen.

4. Implementing CSS for CODEML

CSS has many features. Below we will only discuss the major ones that affect our final CSS to quite a big extent. For full description about all the rules used in the final product one is recommended to check the CSS2 Specification found in its official site at http://www.w3.org/TR/REC-CSS2/cover.html. However below we will discuss some major CSS features that concern our final product to great extent.

4.1 Attaching Style Sheets

We tell the web browser loading the CODEML to apply the style sheet found in the file codeml.css by using the xml-stylesheet processing instruction after the DOCTYPE declaration in the code of listing 1.2 as below

```
        <?xml-stylesheet type="text/css" href="codeml.css"?>
```

The ‘type’ pseudo-attribute is the MIME media type of the style sheet in use and the ‘href’ pseudo-attribute is the URL of the style sheet. The URL is often relative but you may use an absolute one as well.

4.2 Element Selectors in CSS

CSS provides limited abilities to select the elements to which a given rule applies. In many cases element names and lists of element names separated by commas are sufficient. However, in CODEML just applying a rule to an element is not enough. This is because CSS uses very few tags and the tags can have a different visual meaning depending upon where they are located. Example: the ***cpg*** tag which is used for different kinds of grouping is sometimes an inline element and sometimes a block element. Therefore we need to have much more powerful way of selecting the elements. CSS does provide some basic selectors that can be used for this, though they're by no means as powerful as the XPath syntax of XSLT. For visualizing CODEML we need the following kind of selectors the most.

4.2.1 The Universal Selector "*"

The universal selector '*' applies the rule to all the elements for which another more specific rule doesn't exist. We will use this to set default styles for all elements. As specified in the CODEML specification (Section 2.1.2) we will use * to make the default values of styles affecting text rendering to be inherited.

4.2.2 The Attribute Selector "[ ]"

The attribute selector is one of the most important kinds of selection that we will have to do. Many of the CODEML elements can contain some attribute value. Each element could have a different CSS rule depending on its attribute value. Listing 4.2 shows a possible CSS, which uses the attribute selector, for the CODEML snippet of listing 4.1.

Listing 4.1:A small CODEML Snippet

```
<cpo color="green">return1</cpo>
<cpo color="blue">return2</cpo>
<cpo >return3</cpo>
```

Listing 4.1: CSS rules for the snippet shown in listing 4.1

```
cpo {color: #000000} /* default color black */
cpo[color=”green”] {color: #006600} /* #006600 is a greenish color */
cpo[color=”blue”] {color: #000099}  /* #000099 is a bluish color */
```

4.2.3 The ID Selector “#”

Many of the CODEML tags have an ID attribute. We will use the X#M syntax for selecting this element where it matches all elements named X which is identified with the ID M.

4.2.4 The Child Selector “>”

We will use the syntax X >Y to match all elements Y which is the child of X.

4.3 CSS Properties

CSS provides many properties that can be assigned to an element. Most of these properties affect the layout of the particular element.

4.3.1 Display Property:

From the perspective of CSS, every element is either a block element or an inline element or table parts or invisible. The display property specifies which one of these an element is. Block level elements are laid out vertically, one on top of the other, where as inline elements are laid out horizontally from left to right. In CODEML all other element except ***pcode*** and ***cpg*** are clearly inline elements. However ***pcode*** is always a block element and ***cpg*** can either be inline or block. For example a ***cpg*** which has an attribute ***breakO*** or ***Obreak*** should be a block where as the one which is grouping the content of the same line needs to be inline.

4.3.2 Content Property:

The content property can be used to put data from the style sheet into the output document at a position indicated by *:before* or *:after* pseudo element. This property for CODEML is a highly important. For example let us take a ***cpg*** element with the attribute open and close **<*cpg open=”{“ close=”}”*>**. When ever we see this kind of tag we have to put a “{“ at the beginning of the tag and a “}” at the end of the tag. To do this we can use the content property. Furthermore since we don’t know the value of the attribute open and close in advance we will use the *attr(x)* syntax to generate the attribute value as content. For example in our example we will have the following CSS rule.

```
cpg[open][close]:before{content: attr(open);}
cpg[open][close]:after{content: attr(close);}
```

4.3.3 Padding Properties:

Padding properties specify the amount of space on the inside of the border of the box. We will use padding properties to simulate the effect of text indentation. Since there will be box inside box kind of mechanism for display we can catch the effect of remembering the last indentation. This gives us the possibility to remember the indentation at the upper level and created the new indentation relative to it.

4.4.4 Length Values:

We will use relative units for specifying length value. This will help to improve the display over various screen sizes. One of the most common way to do a relative units measurement is to use *em* and *ex*. *em* is the width of the letter m in the current font and *ex* is the height of the letter x in the current font.

5. CSS and the modern Browser:

CSS is supported by most modern browser like Opera, Firefox, Mozilla, Internet Explorer. However not all of CSS specifications are fully implemented in all of them. The one that supports CSS2 to the greatest extent till today is Opera. Firefox also supports most of the specifications. For our concern the important issue is that the browser needs to support *before* and *after* pseudo-element together with the *content* property. Internet Explorer 6 doesn't support these, so we cannot use CSS to visualize CODEML in Internet Explorer. Firefox supports pseudo elements and content generation but the problem is that it is not possible to select the content generated by CSS. So in situation where one tries to cut and paste the code from the browser he will miss the codes generated by CSS. Another issue is that Firefox has a bug that it cannot create a carriage return in the content generated. In CSS this would be possible by putting *"\A"* in the string literal. So only browser among the four mentioned earlier that support the content generation from CSS correctly is Opera.

6. CSS and CODEML

CSS has very limited tools to select the elements. Moreover it is quite weak in selecting elements with lots of attributes. Where as CODEML is just the opposite in the sense that it uses very few elements but a lot of attributes to catch the meaning of the context. The same ***cpg*** tag can group a whole procedure, a line, an expression and combination of all of this. From the XML point of view this is all possible through the use of various attributes. However to catch all this in CSS is quite difficult. Below is a context where CSS doesn't seem to be enough for CODEML. We will reprint the code snippets we gave earlier in the introduction to CODEML section in the next page.
Now we can see that content of the **<*cpg*>** tag at line 14 to 22 in listing 6.1 is a block level element. Now how do we tell this from CSS. Clearly we cannot simply announce all ***cpg*** as block level. One possible way to capture this is to actually read if the next ***cpg*** element has a ***breakO*** or ***Obreak*** attribute or not. But in CSS this is not possible as you can never select the parent of something. You can only select a child of something.
Another problem encountered was the combination of CSS selectors. Simply put the following doesn't work at all in CSS though it seems from the first glance that it should work:

```
cpg[breakO]:first-child
```

Here we trying to access the first child of all the ***cpg*** element that has attribute ***breakO***. A selection like this has to be used quite often to select the visual meanings in the current format of CODEML. To achieve the look of line 3 in Listing 6.1 we have to include the open symbol "{" from the ***cpg*** of line 23 after the content of the ***cpg*** at line 14. This means we have to make the ***cpg*** of line 18 an inline element but clearly the content inside this ***cpg*** element should be treated as one block.

Listing 6.1: A java code snippet [Koh01]

```
public int find (int x) {
    if (s [x] <0) return x;
    return find (s [x ]);
    }
```

Listing 6.2: The CODEML representation for the code snippet in Listing 1 [Koh01]

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE code SYSTEM “../../dtd/codeml.dtd">
<cpg>
      <cpg>
            <cpo>public</cpo>
            <cptype>int</cptype>
            <cpo>find</cpo>
            <cpg open="(" close=")">
                  <cptype>int</cptype>
                  <cpi>x</cpi>
            </cpg>
      </cpg>
      <cpg open="{" close="}" breakO="hard">
            <cpo>if</cpo>
            <cpg open="(" close=")">
                  <cpo>s</cpo>
                  <cpg open="[" close="]">
                        <cpi>x</cpi>
                  </cpg>
                  <cpo>&lt;</cpo>
                  <cpb type="number">0</cpb>
            </cpg>
            <cpg close=";" Cbreak="hard">
                  <cpo>return</cpo>
                  <cpi>x</cpi>
            </cpg>
            <cpg close=";" Cbreak="hard">
                  <cpo>return</cpo>
                  <cpg>
                        <cpo>find</cpo>
                        <cpg open="(" close=")">
                              <cpo>s</cpo>
                              <cpg open="[" close="]">
                                    <cpi>x</cpi>
                              </cpg>
                        </cpg>
                  </cpg>
            </cpg>
      </cpg>
</cpg>
```

Listing 7.1: The changed CodeML representation of listing 6.2

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE code SYSTEM “../../dtd/codeml.dtd">
<cpg>
      <cpg>
            <cpo>public</cpo>
            <cptype>int</cptype>
            <cpo>find</cpo>
            <cpg open="(" close=")">
                  <cptype>int</cptype>
                  <cpi>x</cpi>
            </cpg>
      </cpg>
      <cpg open="{" close="}" break0="hard">
            <blk indent="1">
                  <cpo>if</cpo>
                  <cpg open="(" close=")">
                        <cpo>s</cpo>
                        <cpg open="[" close="]">
                              <cpi>x</cpi>
                        </cpg>
                        <cpo>&lt;</cpo>
                        <cpb type="number">0</cpb>
                  </cpg>
                  <cpg close=";" Cbreak="hard">
                        <blk indent="0">
                              <cpo>return</cpo>
                              <cpi>x</cpi>
                  </cpg>
            </blk>
         <cpg close=";" Cbreak="hard">
                  <cpo>return</cpo>
                  <blk indent="0">
                        <cpg>
                              <cpo>find</cpo>
                              <cpg open="(" close=")">
                                    <cpo>s</cpo>
                                    <cpg open="[" close="]">
                                          <cpi>x</cpi>
                                    </cpg>
                              </cpg>
                        </cpg>
                  </blk>
            </cpg>
      </blk>
    </cpg>
</cpg>
```

7. Proposed Changes to the CODEML format

As shown in the previous section it is difficult to decide the display type of the ***cpg*** element from the current CODEML format. This can be solved by actually creating a new element that bounds all the elements belonging to one block. That is we actually add a tag ***blk*** to group all the elements belonging to one block. Let's look at the example in listing 6.1 from the view point of display blocks. First the whole code is one big box. Inside this box lies a new box for the code in line 3. Then there should be another box below that contains line 4, 5 and 6. This box has to be indented by some amount. Inside this box contains three similar kinds of blocks for line 4, 5 and 6 each. We will represent each of this block inside an element called ***blk***. The ***blk*** element will have attribute ***indent*** to determine the amount of indent. Indent can have values of 0 to 3 where 0 indicates no indent 1 indicates indent of 1 tab and 2 indicates indent of 2 tabs and so on. For the existing implementation of CODEML one can simply add this new element right after the ***cpg*** element with ***breakO***, ***Obreak***, ***breakC*** and ***Cbreak*** attributes. Listing 7.1 shows the changed CODEML of Listing 6.2.

8. Spacing Error with Browsers

In markup languages the spaces between the two markups are irrelevant. In context of CODEML the following two snippets of codes are same.

i. <cpg><cpo>

ii. <cpg>
 <cpo>

We would expect the browser to behave same when displaying code shown in i. or ii. However browsers like FireFox and Opera behave different for the two codes. They seem to read one extra space for the code ii. This creates unwanted spaces in our display. Looking at listing 6.2 at line 25 and 26 we do not want any space between '(' and 's' however the carriage return after line 25 creates an unwanted space in the display. To avoid this we will have to remove the carriage return at the end of line 25. Because of this bug (which seems to have been put in purpose by the browsers) we will have to remove the carriage return between the elements if we want no space between them. The problem is demonstrated in Appendix B.

9. Evaluation of CSS visualization

The CSS file used to visualize the CODEML is listed in Appendix A. Using this file we were able to achieve a simple visualization. Please refer to Appendix B for output produced for example CODEML files. It is clear that CSS can be used to visualize CODEML. However it is not so rich in formatting. Remember the whole purpose of CODEML is to make the presentation of program codes in the web easy and nice. Also remember we modified the original CODEML for CSS slightly. The presented display is very static. That is it does not interact with the user. In case of program codes a very useful feature would be the ability to expand and collapse a block (a loop, a procedure and so on) of code like the one implemented in Microsoft Visual Studio.Net and Emacs. When the program code is large this feature is very useful to make the task of understanding the code easily. It would be very nice if this feature could be implemented while presenting CODEML. To do this clearly either CSS3 or Javascript has to be used. However there is no support yet for CSS3 and Javascript cannot be included in xml. Quoting from http://www.w3schools.com/xml/xml_display.asp "*Formatting XML with CSS is NOT the future of how to style XML documents. XML document should be styled by using the W3C's XSL standard!*" Today browsers like Mozilla, Firefox and Internet Explorer support XSL. So actually one can just present the user with the XML and the XSL file and the above browsers can apply the XSL to the XML file before presenting to the user. XSL along with XPATH is really rich and powerful. Using XSL we can change our CODEML into html markup and then present it to the user while using CSS to style the HTML. This way we can also include Javascript for content interaction. Thus we get much powerful and rich visualization. In the next sections we will try to explore the CODEML visualization using the combination of XSL, CSS and JavaScript. Note that the XSL based solution is still going to be a client side solution. Since user browsers today are mostly either Mozilla based or Internet Explorer the XSL solution will also cover bigger percentage of users. All these reasons are a very good motivation for using this method than just plain CSS.

10. Visualization using XSL, CSS and JavaScript

First we convert the CODEML mark-up into HTML using XSL. Inside this HTML we embed a CSS file and a script file from some URL. So when the browser is displaying the HTML it actually applies the CSS and the script as well. Appendix C shows all the XSL, CSS and JavaScript file that was created for the visualization. Please not that we do not use the XSL-Fo extension instead we use CSS.

As XPath is very powerful in selecting element the complication that we faced in CSS earlier can be easily solved. In fact we do not need to introduce the new ***blk*** tag. The original CODEML format can be easily visualized. With the help of JavaScript we manage to create the expand and collapse function for the code blocks. The method used is that each block of code is wrapped inside a <ul> tag. Then javascript's onclick property is used to change the display of the <ul> to none or block to show or hide the block, thus simulating the effect of expand and collapse. Appendix D shows the test of the XSL based visualization for example CODEML snippets.

11. Conclusion:

The aim of this project was to create a CSS visualization for CODEML. We showed that with the current specification of CODEML it is quite difficult to achieve the proper display. However we were able to properly display CODEML by just adding one new element **<*blk*>** that groups other elements into blocks. To achieve even better control over the display we went further and tried another solution based on the combination of XSL, CSS and JavaScript. It gave us a much better result while still being client side. This method also covers more users as it works very well with Mozilla, FireFox and Internet Explorer. Since the second method is much more powerful and rich I would conclude that it is the better alternative than just plain CSS. In future the browser support for CSS is going to change. Once there will be support for CSS3 we can completely rely in CSS but until then we may have to use the XSL and CSS combination as it is more complete.

12. Bibliography:

[Koh01]	Michael Kohlhase. CODEML: An Open Markup Format for the Content and Presentation of Program Code: Dec 2004
[Har01]	Elliote Rusty Harold *XML Bible* (New York: Hungry Minds, 2001)
[Har02]	Elliote Rusty Harold, W. Scott Means, *XML in a Nutshell* (New York: O'Reilly, June 2002)
[XPath]	World Wide Web Consortium. *XML Path Language*. W3C Recommendation. See http://www.w3.org/TR/xpath
[XML]	World Wide Web Consortium. *Extensible Markup Language (XML) 1.0*. W3C Recommendation. See http://www.w3.org/TR/REC-xml
[CSS]	World Wide Web Consortium. *Cascading Style Sheets, level 2 (CSS2)*. W3C Recommendation. See http://www.w3.org/TR/REC-CSS2
[XSL]	World Wide Web Consortium. *XSL Transformations (XSLT)*. W3C Recommendation. See http://www.w3.org/TR/xslt

13. Appendix

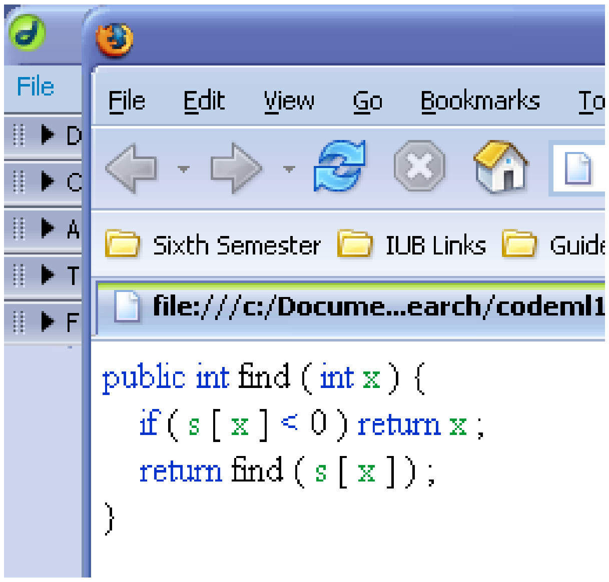
13.1 Appendix A

CSS file for Visualizing CODEML

```
/*define each pcode as a block*/
pcode{
display:block;
padding:5px;
}
/* Write the closing symbol from the attribute*/
cpg[close]:after{
      color: #000000;
      content: attr(close);
}
/* Write the opening symbol from the attribute*/
cpg[open]:before{
      color: #000000;
      content: attr(open);
}
/* Selects Only the '{' attr with Obreak present*/
cpg[breakO="hard"][open="{"]:before{
      color: #000000;
      content: attr(open);
}
/* Selects Only the '{' attr with Obreak present*/
cpg[Obreak="hard"][open="{"]:before{
      color: #000000;
      content: "\A" attr(open) ;
}
/*For the end of each line with breakC attribute*/
cpg[breakC="hard"][close]:after{
      color: #000000;
      content:attr(close) ;
}
/*For the end of each line with Cbreak attribute*/
cpg[Cbreak="hard"][close]:after{
      color: #000000;
      content: attr(close);
}
/*Define elements inside brk as blok and also use the indent
attribute*/
brk[indent="1"] {
      display:block;
      color:#FF3333;
      padding-left:1em;
}
/*Define elements inside brk as blok and also use the indent
attribute*/
brk[indent="2"] {
      display:block;
      color:#FF3333;
      padding-left:2em;
}
/*Define elements inside brk as blok and also use the indent
attribute*/
brk[indent="3"] {
      display:block;
      color:#FF3333;
      padding-left:3em;
}
/*Define elements inside brk as blok and also use the indent
attribute*/
brk[indent="0"] {
      display:block;
      color:#FF3333;
      padding-left:0px;
}
/*blue color for built in words*/
cptype,cpo[type="built-in"] {
      color:#0033CC;
}
/*black color for user defined words*/
cpb,cpo[type="imported"],cpo[type="defined"] {
      color: #000000;
}
/*green color for variables*/
cpi{
      color:#009933;}
/*Grey color for comments and descriptive text*/
cpt{
      color:#999999;}
```

13.2 Appendix B

Here we will show some examples of visualized CODEML in FireFox .
Below is the cropped image of the output produced when the CSS file is attached to the CODEML listed in Listing 7.1 in Section 7.


File
File Edit View Go Bookmarks To
Sixth Semester IUB Links Guide
file:///c:/Docume...earch/codeml1

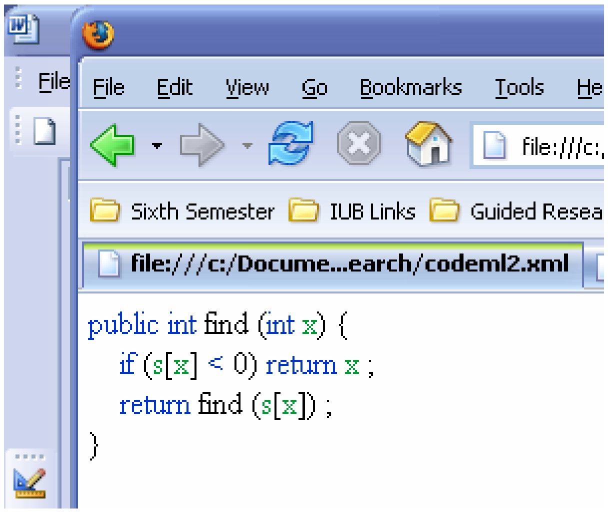
```
public int find ( int x ) {
   if ( s [ x ] < 0 ) return x ;
   return find ( s [ x ] ) ;
}
```

Please note the spaces after the brackets. These were created because the browser puts a space when it sees a carriage return between the two elements. Now we arrange the same source without the Carriage Return as shown below and display it in FireFox.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/css" href="style.css"?>
<pcode>
<cpg>
      <cpg>
            <cpo type="built-in">public</cpo>
            <cptype>int</cptype>
            <cpo>find</cpo>
            <cpg open="(" close=")"><cptype>int</cptype>
                  <cpi>x</cpi></cpg>
      </cpg>
      <cpg open="{" close="}" breakO="harÕ">
            <brk indent="1">
                  <brk indent="0">
                        <cpo type="built-in">if</cpo>
                        <cpg open="(" close=")"><cpi>s</cpi><cpg
open="[" close="]"><cpi>x</cpi></cpg>
                              <cpo type="built-in">&lt;</cpo>
                              <cpb type="number">0</cpb></cpg>
                        <cpg close=";" Cbreak="harÕ">
                              <cpo type="built-in">return</cpo>
```

```
                                    <cpi>x</cpi>
                              </cpg>
                        </brk>
                        <brk indent="0">
                              <cpg close=";" Cbreak="harÕ">
                                    <cpo type="built-in">return</cpo>
                                    <cpg>
                                          <cpo type="defined">find</cpo>
                                          <cpg open="("
close=")"><cpi>s</cpi><cpg open="[" close="]"><cpi>x</cpi></cpg></cpg>
                                    </cpg>
                              </cpg>
                        </brk>
                  </brk>
            </cpg>
</cpg>
</pcode>
```

Below is the screenshot of the display:


File
File Edit View Go Bookmarks Tools He
file:///c:,
Sixth Semester
IUB Links
Guided Resea
file:///c:/Docume...earch/codeml2.xml

```
public int find (int x) {
    if (s[x] < 0) return x ;
    return find (s[x]) ;
}
```

Notice that the spaces are gone and the result is much more desired one.

13.3 Appendix C

Here we will list all the 3 different source files for XSL, CSS and JavaScript that was produced to visualize CODEML with the feature of expandable and collapsible code blocks.

The XSL codes:

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" indent="yes" doctype-public="-//W3C//DTD HTML
4.01//EN"
      doctype-system="http://www.w3.org/TR/html4/strict.dtd"/>
      <xsl:template match="/">
        <html>
          <head>
            <title>CodeML</title>
            <script type='text/javascript'>
            window.onload = function(){
              if (window.winOnLoad) window.winOnLoad();
            }
            </script>
            <link rel='stylesheet' type='text/css' href='codeml.css'/>
            <script type='text/javascript' src='codeml.js'></script>
            </head>
            <body>
              <xsl:apply-templates/>
            </body>
        </html>
      </xsl:template>
      <xsl:template match="pcode">
        <ul id='pcode1' class='pCode'>
          <li>
            <xsl:for-each select="*">
              <xsl:call-template name="whole" />
            </xsl:for-each>
          </li>
        </ul>
      </xsl:template>
      <xsl:template name="whole">
        <xsl:if test="./cpg[1]">
          <span>
            <xsl:call-template name="break"/>
              <xsl:if test="./cpg[2]/@breakO">
                <xsl:value-of select="./cpg[2]/@open"/>
              </xsl:if>
          </span>
        </xsl:if>
        <xsl:if test="./cpg[2]">
          <ul>
            <xsl:if test="./cpg[2]/@Obreak">
              <li>
                <xsl:value-of select="./cpg[2]/@open"/>
              </li>
            </xsl:if>
            <xsl:for-each select="./cpg[2]/*">
              <xsl:if test="@Cbreak">
                <li>
                  <xsl:choose>
                    <xsl:when test="self::cpg">
                      <xsl:for-each select="*">
                        <xsl:if test="@open">
                          <xsl:value-of select="@open"/>
                        </xsl:if>
                        <xsl:call-template name="cpgsecond"/>
                        <xsl:if test="@close">
                          <xsl:value-of select="@close"/>
                        </xsl:if>
                      </xsl:for-each>
                    </xsl:when>
                  </xsl:choose>
                  <xsl:value-of select="@close"/>
                </li>
              </xsl:if>
              <xsl:if test="@Pbreak">
                <li>
                  <xsl:call-template name="whole"/>
                </li>
               </xsl:if>
            </xsl:for-each>
              <li>
                <xsl:value-of select="./cpg[2]/@close"/>
              </li>
            </ul>
          </xsl:if>
      </xsl:template>
      <xsl:template name="break">
        <xsl:for-each select="./cpg[1]/*">
          <xsl:if test="@open">
            <xsl:value-of select="@open"/>
          </xsl:if>
          <xsl:call-template name="cpgsecond"/>
          <xsl:if test="@close">
            <xsl:value-of select="@close"/>
          </xsl:if>
        </xsl:for-each>
      </xsl:template>
      <xsl:template name="cpgsecond">
        <xsl:choose>
          <xsl:when test="self::cpo">
            <xsl:choose>
              <xsl:when test="./@type='built-in'">
                <div class="cpobuiltin">
                  <xsl:value-of select="."/>
                  <xsl:text> </xsl:text>
                </div>
              </xsl:when>
              <xsl:otherwise>
                <div class="cpo">
                  <xsl:value-of select="."/>
```

```
                  <xsl:text> </xsl:text>
                </div>
              </xsl:otherwise>
            </xsl:choose>
          </xsl:when>
          <xsl:when test="self::cpb">
            <div class="cpb"><xsl:value-of select="."/></div>
          </xsl:when>
          <xsl:when test="self::cpi">
            <div class="cpi"><xsl:value-of select="."/></div>
          </xsl:when>
          <xsl:when test="self::cptype">
            <div class="cptype">
              <xsl:value-of select="."/>
              <xsl:text> </xsl:text>
            </div>
          </xsl:when>
          <xsl:when test="self::cpg">
            <xsl:for-each select="*">
              <xsl:if test="@open">
                <xsl:value-of select="@open"/>
              </xsl:if>
              <xsl:call-template name="cpgsecond"/>
              <xsl:if test="@close">
                <xsl:value-of select="@close"/>
              </xsl:if>
            </xsl:for-each>
          </xsl:when>
        </xsl:choose>
      </xsl:template>
      <xsl:template match="*">
      </xsl:template>
</xsl:stylesheet>
```

The JavaScript Codes:

```
var mnu = new Array();
function winOnLoad()
{
  mnu[0] = new BlockCollapse('pcode1');
 }
window.onunload = function(){
  return;
}
function BlockCollapse(parentElement, action){
  var container = document.getElementById(parentElement);
  if (!container) {return null;}
  var isUL = container.nodeName.toUpperCase() == 'UL';
  var i, target, aTgt = document.getElementsByTagName(isUL ?
'UL':'DIV', container);
  for (i = 0; i < aTgt.length; ++i) {
    target = PreviousElement(aTgt[i]);
    if (target && (isUL || target.nodeName.charAt(0).toUpperCase() ==
'H')) {
      aTgt[i].style.display = action ? 'block' : 'block';
      target.style.cursor = 'default';
      target.TgtCode = aTgt[i];
      target.onclick = trg_onClick;
    }
  }
  function trg_onClick()  {
    var tgt = this.TgtCode.style;
      var pu = this.parentNode;
      var spa=pu.firstChild.style;
    if (tgt.display == 'block') {
      tgt.display = 'none';
        spa.backgroundImage='url(plus.gif)';
    }
    else {
      tgt.display = 'block';
        spa.backgroundImage='url(minus.gif)';
    }
  }
  this.onUnload = function()  {
    if (!container || !aTgt) {return;}
    for (i = 0; i < aTgt.length; ++i) {
      target = PreviousElement(aTgt[i]);
      if (target && (isUL || target.nodeName.charAt(0).toUpperCase ==
'H')) {
        target.TgtCode = null;
        target.onclick = null;
      }
    }
  }
  function PreviousElement(element, tag){
    var s = element ? element.previousSibling : null;
    if (tag) while (s && s.nodeName != tag) { s = s.previousSibling; }
    else while (s && s.nodeType != 1) { s = s.previousSibling; }
    return s;
  }
}
```

The CSS Codes:

```
body {
      background:#eef;
      font-family: Verdana, arial, serif;
      font-size:12px;
      }
.pCode {
      margin:0;
      padding:0;}
div {
      display:inline;
}
#pcode1 ul {
      margin:0; padding:0;
      background:#eef;
      background-image:url(ul.gif);
      background-repeat:repeat-y;
```

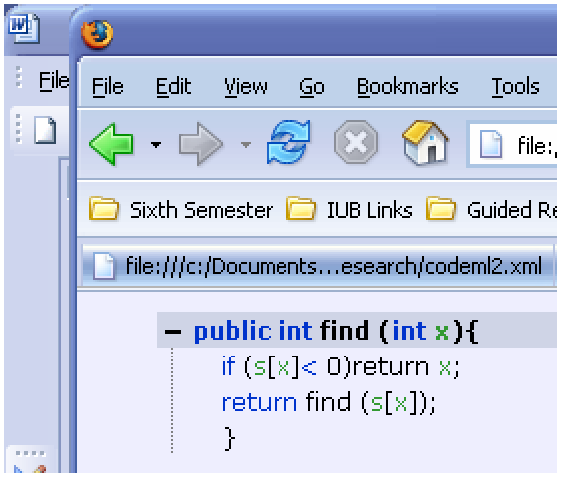
```
 }
#pcode1 li {
      margin:0px 0 0px 0;
      padding:2px 2px 2px 32px;
      list-style-type:none;
}
#pcode1 span {
      display:block;
      font-weight:bold;
      padding:2px;
      padding-left:18px;
      background:#CFD4E6;
      background-image:url(minus.gif);
      background-repeat:no-repeat;
}
.cpo {
      color:#000000;
}
.cptype,.cpobuiltin{
      color:#0033CC;
}
.cpi{
      color: #339933;
}
```

13.4 Appendix D

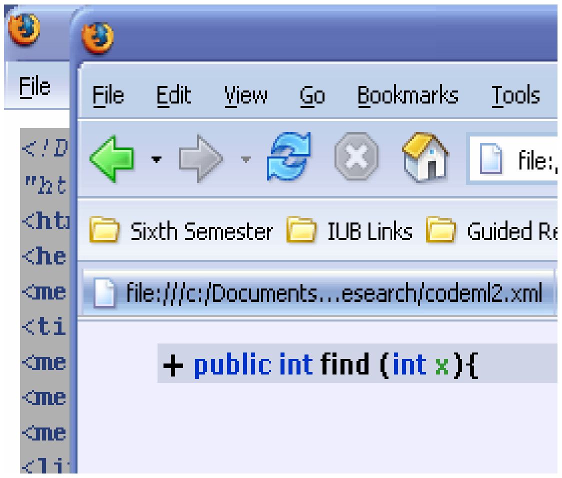
Here we will show the result of the XSL styling of CODEML. A CODEML file as below was opened in Firefox:

```
<?xml version="1.0" encoding="utf-8" ?>
<?xml-stylesheet type="text/xml" href="codeml.xsl"?>
<pcode format="java">
  <cpg>
      <cpg>
            <cpo type="built-in">public</cpo>
            <cptype>int</cptype>
            <cpo>find</cpo>
            <cpg open="(" close=")">
                  <cptype>int</cptype>
                  <cpi>x</cpi>
            </cpg>
      </cpg>
      <cpg open="{" close="}" breakO="hard" indent="1">
            <cpg close=";" Cbreak="hard">
                  <cpo type="built-in">if</cpo>
                  <cpg open="(" close=")">
                        <cpi>s</cpi>
                        <cpg open="[" close="]">
                              <cpi>x</cpi>
                        </cpg>
                        <cpo type="built-in">&lt;</cpo>
                        <cpb type="number">0</cpb>
                  </cpg>
                  <cpo>return</cpo>
                  <cpi>x</cpi>
            </cpg>
            <cpg close=";" Cbreak="hard">
                  <cpo type="built-in">return</cpo>
                  <cpg>
                        <cpo>find</cpo>
                        <cpg open="(" close=")">
                              <cpi>s</cpi>
                              <cpg open="[" close="]">
                                    <cpi>x</cpi>
                              </cpg>
                        </cpg>
                  </cpg>
            </cpg>
      </cpg>
   </cpg>
</pcode>
```

Below are the screenshot of the display:


File Edit View Go Bookmarks Tools
Sixth Semester IUB Links Guided Re
file:///c:/Documents...esearch/codeml2.xml

```
– public int find (int x){
    if (s[x]< 0)return x;
    return find (s[x]);
    }
```


File Edit View Go Bookmarks Tools
Sixth Semester IUB Links Guided Re
file:///c:/Documents...esearch/codeml2.xml

```
+ public int find (int x){
```

As the picture shows the procedure find can be expanded or collapsed from the screen.