CODEML: An Open Markup Format the Content and Presentation of Program Code

Michael Kohlhase
Computer Science, Carnegie Mellon University
Pittsburgh, Pa 15213, USA
http://www.cs.cmu.edu/~kohlhase

February 15, 2008
Abstract

This specification defines the Code Markup Language, or CODEML. CODEML is an XML application for describing program code and capturing both its structure and content. The goal of CODEML is to enable program code to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text.

CODEML is inspired by the MathML and OpenMath format for markup of mathematical formulae and can be used in conjunction with these formats in a variety of document formats.

Status of this Document: This specification of the markup language CODEML is intended primarily for a readership consisting of those who will be developing or implementing renderers or editors using it, or software that will communicate using CODEML as a protocol for input or output. It is not a User’s Guide but rather a reference document.

The current report is an early draft of the specification of the CODEML format. Even though the basic layout of the format should survive, details like element- or attribute names, and content models are very likely to change as implementation and application experience grows.
Contents

1 Introduction 2

2 CodeML Elements 4
   2.1 Presentation Elements 4
      2.1.1 Token Elements 5
      2.1.2 Token Element Attributes 7
      2.1.3 Grouping, Indenting and Breaking 9
      2.1.4 Descriptive Text and Comments 10
      2.1.5 Raw Code 11
      2.1.6 Style Change 13
      2.2 Content Elements 14
      2.2.1 Token Elements 14
      2.2.2 Complex Elements 16
   2.3 Interaction of Content and Presentation 17
      2.3.1 Parallel Markup by Cross-references 19

3 Conclusion 22

A Quick-Reference Table to the CodeML Elements 28

B Quick-Reference Table to the CodeML Attributes 29

C The CodeML RelaxNG Schema 32
   C.1 The CODEML Schema Driver 32
   C.2 The CODEML Module PRES 33
   C.3 The CODEML Module CONT 34
   C.4 The CODEML Module INTER 35
      C.4.1 The Dublin Core Schema 36
      C.4.2 The MARC Relators Schema 37
Chapter 1

Introduction

This specification defines the Code Markup Language, or CODEML. CODEML is an XML application for describing program code and capturing both its structure and content. The goal of CODEML is to enable program code to be served, received, and processed on the World Wide Web, just as HTML has enabled this functionality for text.

The primary purpose of program code is as a means of communication between the programmer and the machine as a vehicle to express programs that can be executed by the machine. For this purpose, the field of computer science has developed a multitude of programming languages, i.e. formal languages with a well-defined structure and fixed semantics that allow the programmer to express her operational intentions as texts called program code. This code is then interpreted by a suitable program (e.g. a compiler) and translated into native instructions to the computer, which are then executed to achieve the desired result.

An auxiliary purpose of program code is to communicate about algorithms and programs between humans, e.g. in program development documentation, quality assurance, or computer science education. To allow humans to cope with the complexities of program code, the program development process is supported by a variety of “intelligent development environments”, which provide features like code pretty-printing (e.g. syntax highlighting, indentation), hyper-linking, and even higher-level techniques like literate programming, or documentation extraction from commented code.

Unfortunately, this functionality is directly tied to the development environment and (usually) to a particular programming language: The program text has to be analyzed, and the structural and syntactic information has
to be conveyed to the user in a dedicated user interface. It is currently only possible to export the functionality of these development environments to the World Wide Web to a very limited extent.

- Web markup formats like HTML only provide limited primitives for pretty-printing program listings; the possibilities are largely limited to using the `pre` element, tables, or non-breaking spaces `\&nbsp;`. Together with CSS styling this is used for (static, fixed-width) indenting and syntax highlighting.

- Using technologies like XSL formatting objects (which allow flexible-width formatting\(^1\)) is problematic, since the syntactic and structural information necessary for pretty-printing is implicit in the program structure and a full parser that is needed to reveal it, transcends the possibilities of both `javascript` and `XSLT` (which are the computational engines built into current browsers). Moreover, an approach based on parsing programming languages would necessitate the availability of grammars for all programming languages.

- As programming languages are optimized for compilation and execution, they do not supply an infrastructure for referencing program fragments, which would be\(^2\)

The concrete design of the CODEML format is inspired by the MATHML (the Mathematical Markup Language) and OPENMATH format for markup of mathematical formulae and can be used in conjunction with these formats in a variety of document formats.

The most relevant related work is probably the `listings.sty` package [?](http://www.math.utk.edu/~hildebrand/texdoc/) for `LaTeX`, which allows to mark up program listings for print presentation.\(^4\)

\(^1\)EdNote: check that they do
\(^2\)EdNote: continue
\(^3\)EdNote: say something more, dynamics, local services without program interpretation.
\(^4\)EdNote: say something more, maybe move somewhere else
\(^5\)EdNote: given an overview over what there already is.
Chapter 2

CodeML Elements

The CodeML format is loosely modeled after the MathML format [?]
format. It has three kinds of elements, which we will specify in the rest of the
section: presentation elements (section 2.1), content elements (section 2.2),
and integration elements (section 2.3).

Since the CodeML fragment is intended as a module for the OMDoc [?]
format, it only needs to be concerned with representing the actual code and
can re-use the library- and theory levels of OMDoc6.

The CodeML presentation elements are used to mark up the syntactic
structure of program source code. They classify and group sub-strings of
code in arbitrary programming languages by their syntactic function. As
a consequence, generic tools can perform added-value services, without re-
quiring parsers for the code.7

2.1 Presentation Elements

In this section we will introduce the CodeML presentation elements, which
can be used to classify and group sub-strings of program code by their
syntactic function.

All presentation CodeML elements have the following attributes in com-
mon:

\texttt{xlink:xref} This optional attribute specifies a cross-reference to a content-
CodeML element that corresponds to the code contained in this el-
ment. This attribute (together with the next two) conforms to the
Listing 2.1: A Java code snippet

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

W3C XLink recommendation [?] and is recognized by many XML applications.

**xlink:type** This attribute must have the value 'simple', according to the XLink recommendation. It specifies that xlink:xref is a so-called simple link, like the a element from HTML. The DTD (see appendix ??) fixes this attribute, so in CodeML documents with DTD, it need not be explicitly specified.

**xlink:role** This attribute specifies the role of link established by the xlink:xref as a semantic equivalence. Like the xlink:type, the DTD fixes its value.

We will use the code fragment for a Java implementation of the find procedure in Listing 2.1 as a running example in this section. The presentation-CodeML equivalent of this presented in Listing 2.2.

### 2.1.1 Token Elements

Token elements in presentation markup are broadly intended to represent the smallest code fragments which carry meaning. Thus character data in CodeML markup is only allowed to occur as part of the content of token elements. The only exception is whitespace between elements, which is ignored. Token elements can contain any sequence of zero or more Unicode characters.

Token elements should be rendered as their content (i.e. in the visual case, as a closely-spaced horizontal row of standard glyphs for the characters in their content). Rendering algorithms should also take into account the style attributes as described below, and modify surrounding spacing by rules or attributes specific to each type of token element.

CodeML has the following three elements to classify token elements; all of these can contain arbitrary unicode text\(^1\)

\(^1\)and the XML elements specified by changing the &cpt.extra.content; parameter entity either in the internal subset of the DOCTYPE declaration or in the DTD driver
Listing 2.2: The CODEML presentation for the code snippet in Listing 2.1

1  <cpg>
  <cpo>public</cpo><cptype>int</cptype><cpo>find</cpo>
  <cpg open="("
  <cpg close=")"
  <cptype>int</cptype><cpi>x</cpi></cpg>
</cpg>
6  <cpg open="{" close="}"
  breakO="hard">
  <cpo>if</cpo>
  <cpg open="("
  <cpg close=")"
  <cpi>x</cpi></cpg>
  <cpg open="["
  <cpg close="]"
  <cpi>x</cpi></cpg>
11  <cpb type="number">0</cpb>
</cpg>
<cpg close=";"
  Cbreak="hard">
<cpo>return</cpo><cpi>x</cpi></cpg>
<cpo>return</cpo>
16  <cpg>
  <cpo>find</cpo>
  <cpg open="("
  <cpg close=")"
  <cpi>x</cpi></cpg>
</cpg>
21  <cpg open="["
  <cpg close="]"
  <cpi>x</cpi></cpg>
</cpg>
</cpg>
</cpg>
cpb for basic objects. This element should be used for values like numbers,\(^8\). It has the optional attribute type, which can take the values 'number'\(^9\) to classify its contents.

cpo for operators, i.e. names of symbols with a fixed meaning. These include the built-in operators of the programming language, functions that are imported from a library, and defined ones. The optional type supplies the values 'built-in', 'imported', and 'defined' for these cases.\(^10\)

cpi for identifiers in the programming language, these are usually variable names\(^11\).

cptype for data types\(^12\).

The classification supported by these elements can be used by rendering engines e.g. to highlight or color program code.

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpb</td>
<td>id, xlink:*, type, variant, style, color, background</td>
<td>CDATA</td>
</tr>
<tr>
<td>cpo</td>
<td>id, xlink:*, type, variant, style, color, background</td>
<td>CDATA</td>
</tr>
<tr>
<td>cpi</td>
<td>id, xlink:*, variant, style, color, background</td>
<td>CDATA</td>
</tr>
<tr>
<td>cptype</td>
<td>id, xlink:*, variant, style, color, background</td>
<td>CDATA</td>
</tr>
</tbody>
</table>

Figure 2.1: The CODEML Token Elements

2.1.2 Token Element Attributes

Until full Unicode support is widely available, CODEML provides four code style attributes (see Figure 2.2). These attributes are valid on all presenta-

---

\(^8\) EdNote: Maybe better call it cpv?

\(^9\) EdNote: what else?

\(^10\) EdNote: are there more? yes: recursive-call and definiens

\(^11\) EdNote: what else?

\(^12\) EdNote: do we want this as a separate thing, or can we subsume them into cpo
tion token elements and on no other elements except cstyle.

<table>
<thead>
<tr>
<th>Name</th>
<th>values</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>variant</td>
<td>'normal', 'bold', 'italic', 'bold-italic', 'double-struck', 'bold-fraktur', 'script', 'bold-script', 'fraktur', 'sans-serif', 'bold-sans-serif', 'sans-serif-italic', 'sans-serif-bold-italic', 'monospace'</td>
<td>normal</td>
</tr>
<tr>
<td>size</td>
<td>'small', 'normal', 'big', 'number', 'v-unit'</td>
<td>inherited</td>
</tr>
<tr>
<td>color,</td>
<td>#rgb', '#rrggbb', 'aqua', 'black', 'blue', 'fuchsia', 'gray', 'green',</td>
<td>inherited</td>
</tr>
<tr>
<td>background</td>
<td>'lime', 'maroon', 'navy', 'olive', 'purple', 'red', 'silver', 'teal',</td>
<td></td>
</tr>
<tr>
<td></td>
<td>'white', 'yellow'</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2: CODEML Style Attributes

The style attributes define logical classes of token elements. Each class is intended to correspond to a collection of typographically-related symbolic tokens that have a meaning within a given math expression, and therefore need to be visually distinguished and protected from inadvertent document-wide style changes which might change their meanings.

When CODEML rendering takes place in an environment where CSS is available, the style attributes can be viewed as predefined selectors for CSS style rules. When CSS is not available, it is up to the internal style mechanism of the rendering application to visually distinguish the different logical classes.

Token elements also permit id, class and style attributes for compatibility with style sheet mechanisms.\(^{13}\)

Since CODEML expressions are often embedded in a textual data format such as XHTML, the surrounding text and the CODEML must share rendering attributes such as font size, so that the renderings will be compatible in style. For this reason, most attribute values affecting text rendering are inherited from the rendering environment, as shown in the ‘default’ column in the table above.

\(^{13}\) EdNote: rethink; put this into DTD
The color attribute controls the color in which the content of tokens is rendered. Additionally, when inherited from cstyle, it controls the color of all other drawing by CODEML elements.

The values of color, and background can be specified as as an HTML color name\(^2\) or as a string consisting of ‘#’ followed without intervening whitespace by either 1-digit or 2-digit hexadecimal values for the red, green, and blue components, respectively, of the desired color, with the same number of digits used for each component (or as the keyword ‘transparent’ for background). The hexadecimal digits are not case-sensitive. The possible 1-digit values range from 0 (component not present) to F (component fully present), and the possible 2-digit values range from 00 (component not present) to FF (component fully present), with the 1-digit value x being equivalent to the 2-digit value xx (rather than x0).

The color syntax described above is a subset of the syntax of the color and background-color properties of CSS. The background-color syntax is in turn a subset of the full CSS background property syntax, which also permits specification of (for example) background images with optional repeats. The more general attribute name background is used in CODEML to facilitate possible extensions to the attribute’s scope in future versions of CODEML.

2.1.3 Grouping, Indenting and Breaking

One of the most important CODEML uses of the information in presentation CODEML markup is to indent and linebreak program code. Conventionally, this is hardcoded by tabstops and spaces in the source code, which is sufficient for program development, but carries implicit assumptions about screen size and output format, which are not sustainable in the Internet and ubiquitous computing age (just imagine debugging lisp code on a palmtop). Therefore CODEML relies on the concept of “semantic” concept of code grouping to convey the syntactical structure of programming language expressions, and relies on style sheets or presentation engine to translate this into visual aids, such as indentation, or folding.

CODEML uses the cpg element to group program code fragments, as a consequence, it can contain any (number of) presentation CODEML elements. Since grouping is often supported by bracketing structures in programming languages, the cpg element has three optional attributes open and close which specify the opening and closing brackets used the presented code. The children of the group are separated by single spaces, unless this

\(^2\)Note that the color name keywords are not case-sensitive, unlike most keywords in CODEML attribute values for compatibility with CSS and HTML.
is overridden by setting the optional separator attribute to something else (e.g. a linefeed). Furthermore, the cpg element has an optional indent attribute, which allows the author to specify a factor for the default indentation increment used by the presentation agent. Finally, the cpg element has six attributes that control the linebreaking. They come in three groups of pairs one each for opening, closing brackets and separators. breakO specifies the break before the opening bracket, and 0break a break after it. Attributes breakC and Cbreak to the analogous for the closing brackets, while breakS and Sbreak address the breaks for the separators. Currently, CODEML does not restrict the values of these attributes leaving the specifics to applications; we suggest using the keyword hard for “hard” (unconditional) breaks. A system of numbers might be used for s.¹⁴

Even if the ultimate linebreaking of the code presentation has to be determined by the presentation agent, CODEML provides the empty cpbr element that allows the author to force a line break even if the presentation agent does not see the need for one on the basis of the grouping information.¹⁵

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>cpg</td>
<td>id, xlink:* , open, close, separator, indent, breaks</td>
<td>(cpg</td>
</tr>
<tr>
<td>cpbr</td>
<td>id, xlink:*</td>
<td>CDATA</td>
</tr>
</tbody>
</table>

Figure 2.3: The CODEML Grouping Elements

2.1.4 Descriptive Text and Comments

Many programs contain descriptive text, either as documentation, comments, or (in the case of pseudo-code) even as part of the “programming language”. CODEML provides the elements cpd for descriptive text and cpc for comments. Both elements can contain a multilingual group of cpt elements that contain the actual natural language text.

cpt elements have an xml:lang attribute that specifies the language they are written in. Thus using multilingual groups of cmp elements with

³Note that we cannot specify a breaking strategy for individual separator positions; all are treated alike.
¹⁴EdNote: give/discuss examples
¹⁵EdNote: do we really need this
different languages can promote CODEML internationalization. Conforming with the XML recommendation, we use the ISO 639 two-letter country codes \((en \equiv \text{English}, \ de \equiv \text{German}, \ fr \equiv \text{French}, \ nl \equiv \text{Dutch}, \ldots)\). This optional attribute has the default “’en’”, so that if no \texttt{xml:lang} is given, then English is assumed. Of course it is forbidden to have more than one \texttt{cpt} per value of \texttt{xml:lang} per \texttt{cpd} or \texttt{cpc} element, moreover, \texttt{cpts} that are siblings must be translations of each other.

Let us consider an example: Listing 2.3 shows some presentation-CODEML for our running example from Listing 2.1. This example is mildly internationalized: in line 10, we have a \texttt{cpd} element and in line 19, we have a comment; both English and German text. A suitable presentation engine could generate localized presentations like the ones in Figure 2.5 from the source in Listing2.3.\footnote{EdNote: can the text contain code snippets again? Update the DTD and the examples to allow for that.}
Listing 2.3: Pseudo-code for the code snippet in Listing 2.1

```pseudo-code
procedure find(x)
    if s[x] is negative
        return
    end if

    if we are at the root
        return
    end if

    way to go!
```
2.1.5 Raw Code

To facilitate migration from raw code and to allow for partial markup, CODEML allows to include\(^{17}\) raw code in the \texttt{cpr} element\(^{18,19}\).

\[\text{Issue(17)}\]
\[\text{Issue(18)}\]
\[\text{EdNote(19)}\]

2.1.6 Style Change

The \texttt{cstyle} element is used to make style changes that affect the rendering of its contents. \texttt{cstyle} can be given any attribute accepted by any CODEML presentation element provided that the attribute value is inherited, computed or has a default value; presentation element attributes whose values are required are not accepted by the \texttt{mstyle} element. In addition \texttt{mstyle} can also be given certain special attributes listed below.

The \texttt{cstyle} element accepts any number of arguments. If this number is not 1, its contents are treated as a single ‘inferred \texttt{cpg}’ formed from all its arguments.

Loosely speaking, the effect of the \texttt{cstyle} element is to change the default value of an attribute for the elements it contains. Style changes work in one of several ways, depending on the way in which default values are specified for an attribute. The cases are:

- Some attributes, such as \texttt{size} or \texttt{color}, are inherited from the surrounding context when they are not explicitly set. Specifying such an attribute on an \texttt{cstyle} element sets the value that will be inherited by its child elements. Unless a child element overrides this inherited value, it will pass it on to its children, and they will pass it to their children, and so on. But if a child element does override it, the new (overriding) value will be passed on to that element’s children, and then to their children, etc, until it is again overridden.

- Other attributes, such as \texttt{variant}, have default values that are not normally inherited. That is, if the \texttt{variant} attribute is not set on an element, it will normally use the default value of ‘normal’, even if it was contained in a larger element that set this attribute to a different value. For attributes like this, specifying a value with an \texttt{cstyle} element has the effect of changing the default value for all elements within its scope. The net effect is that setting the attribute value with \texttt{cstyle} propagates the change to all the elements it contains.

\[\text{\textsuperscript{17}Issue: do we also want to allow this in content?}\]
\[\text{\textsuperscript{18}Issue: collapse this with the \texttt{rawcode} element?}\]
\[\text{\textsuperscript{19}EdNote: table?}\]
directly or indirectly, except for the individual elements on which the value is overridden. Unlike in the case of inherited attributes, elements that explicitly override this attribute have no effect on this attribute’s value in their children.

Note that attribute values inherited from an \texttt{cstyle} in any manner affect a given element in the \texttt{cstyle}’s content only if that attribute is not given a value in that element’s start tag. On any element for which the attribute is set explicitly, the value specified on the start tag overrides the inherited value.

As stated above, \texttt{style} accepts all attributes of all \texttt{CodeML} presentation elements which do not have required values. That is, all attributes which have an explicit default value or a default value which is inherited or computed are accepted by the \texttt{mstyle} element.

\section{Content Elements}

The content fragment of \texttt{CODEML} is used to represent the abstract structure of the algorithm underlying the program and unambiguously identify the symbols used in it, independently of the surface form. We will again use the JAVA code fragment for the \texttt{find} procedure in Listing 2.1 as a running example in this section. The content-\texttt{CODEML} equivalent of this is given in Listing 2.4.

The \texttt{CODEML} content elements only have the common \texttt{CODEML} attributes\footnote{EdNote: crossref, define them somewhere above} attributes, and an optional \texttt{id} attribute, which can be used for cross-referencing by by the \texttt{xlink} attributes by the presentation-\texttt{CODEML} elements (see the introduction to section 2.1 and section 2.3).

\subsection{Token Elements}

The \texttt{CODEML} token elements are similar in nature to the \texttt{OpenMath} elements, we have:

\texttt{ccv} for variables: this element has a required attribute \texttt{name} for the name of the variable. Since for variables the name determines the content, this element is empty.

\texttt{ccsym} for defined symbols: this element is empty as well and has two required attributes \texttt{name} for the name of the symbols, and \texttt{cd} for the
Listing 2.4: The CODEML content for the code snippet in Listing 2.1

```xml
<ccdef export="find">
  <ccsym cd="java.dec" name="public-type-function"/>
  <apply>
    <ccsym cd="java.types" name="funtype"/>
    <ccsym cd="java.types" name="int"/>
    <ccsym cd="java.types" name="int"/>
  </apply>
  <bind>
    <bvar><ccv name="x"/></bvar>
    <apply>
      <ccsym cd="java.control" name="if"/>
      <apply>
        <ccsym cd="java.arith" name="less"/>
        <apply>
          <ccsym cd="java.array" name="select"/>
          <ccv name="s"/>
          <ccv name="x"/>
        </apply>
      </apply>
    </apply>
    <ccb type="number">0</ccb>
    <apply>
      <ccsym cd="java.proc" name="return"/>
      <ccv name="x"/>
    </apply>
    <apply>
      <ccsym cd="java.proc" name="return"/>
      <apply>
        <ccsym cd="union-find" name="find"/>
        <apply>
          <ccsym cd="java.array" name="select"/>
          <ccv name="s"/>
          <ccv name="x"/>
        </apply>
      </apply>
    </apply>
    <apply>
      <ccsym cd="java.proc" name="return"/>
      <ccb type="number">1</ccb>
      <apply>
        <ccsym cd="java.proc" name="return"/>
        <ccv name="x"/>
      </apply>
      <apply>
        <ccsym cd="java.proc" name="return"/>
        <ccv name="x"/>
      </apply>
    </apply>
  </bind>
</ccdef>
```
<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>apply</td>
<td>id</td>
<td>(apply</td>
</tr>
<tr>
<td>bind</td>
<td>id</td>
<td>ccsym, bvar, (apply</td>
</tr>
<tr>
<td>bvar</td>
<td>id</td>
<td>ccv*</td>
</tr>
<tr>
<td>ccv</td>
<td>name</td>
<td>id</td>
</tr>
<tr>
<td>ccysm</td>
<td>cd, name</td>
<td>id</td>
</tr>
<tr>
<td>ccdef</td>
<td>export</td>
<td>id</td>
</tr>
</tbody>
</table>

Figure 2.6: The CodeML Content Elements

that describes, implements, or defines the meaning of the symbol. These two attributes totally determine the content of the symbol (by reference to a defining document\(^{21}\)), so the ccsym element is empty.

ccb that contains basic data, such as numbers, strings, etc. The type of this data can be specified in the optional attribute type\(^{22}\)

### 2.2.2 Complex Elements

Content-CodeML knows the following three complex elements, which allow to combine content-CodeML expressions to larger ones.

apply for (function/procedure) application. This element is a constructor for function- and procedure calls: The first child is interpreted as a function or procedure and is given the remaining children of the apply element as argument.

bind this binding constructor bind is used to represent binding and abstraction constructions. It is primarily used in function and procedure definitions. The first child of this a CodeML binding expression must be a ccsym element, which defines a binding symbol. The second child must be a bvar element, which contains ccv elements for all the formal parameters of the binding. The third child of the bind element is known as the body. It is an arbitrary content-CodeML element,

\(^{21}\)EdNote: say some more about the module system, ... later

\(^{22}\)EdNote: do we want to enumerate the basic data types, or leave them open for the user?
which may (but in general need not) contain occurrences of the bound variables. Note that the bound variables specified in the \texttt{bvar} element only have or in the body. An occurrence of an \texttt{ccv} element outside the body is semantically a different variable, even if it carries the same name.

\texttt{ccdef} for definitions and declarations: In contrast to the \texttt{bind} element, this one defines a set of symbols to be visible outside its body: Its required \texttt{exports} attribute specifies a set of symbol names as a whitespace-separated list of names.

The first child of the \texttt{ccdef} element is a declaration symbol (specified as a \texttt{ccsym} element that specifies the definition schema. For instance, in the \texttt{ccdef} element in Listing 2.4, this is the symbol

\begin{verbatim}
<ccsym cd="java.dec" name="public−type−function"/>
\end{verbatim}

that specifies that the second child is a type (the type of the defined symbol), and the third child is a representation of a function (in this case via a \texttt{bind} element).

\section*{2.3 Interaction of Content and Presentation}

In the last sections we have seen two styles of markup for program code:

\emph{Presentation markup} captures notational structure. It encodes the notational structure of an expression in a sufficiently abstract way to facilitate rendering to various media. It does this by providing information such as structured grouping of expression parts, classification of symbols, etc.

Presentation markup does not directly concern itself with the mathematical structure or meaning of the code. In many situations, notational structure and mathematical structure are closely related, so a sophisticated processing application may be able to heuristically infer mathematical meaning from notational structure, provided sufficient context is known. However, in practice, the inference of mathematical meaning from mathematical notation must often be left to the reader. As a consequence, employing presentation tags alone may limit the ability to re-use a CodeML object in another context, especially evaluation by external applications.

\emph{Content markup} captures mathematical structure. It encodes mathematical structure in a sufficiently regular way in order to facilitate the assignment of mathematical meaning to an expression by application programs.

\footnote{NEW PART: copied from MathML, rework heavily}
Though the details of mapping from mathematical expression structure to mathematical meaning can be extremely complex, in practice, there is wide agreement about the conventional meaning of many basic mathematical constructions. Consequently, much of the meaning of a content expression is easily accessible to a processing application, independently of where or how it is displayed to the reader. In many cases, content markup could be cut from a Web browser and pasted into a mathematical software tool with confidence that sensible values will be computed.

Since content markup is not directly concerned with how an expression is displayed, a renderer must infer how an expression should be presented to a reader. While a sufficiently sophisticated renderer and style sheet mechanism could in principle allow a user to read mathematical documents using personalized notational preferences, in practice, rendering content expressions with notational nuances may still require intervention of some sort. As a consequence, employing content tags alone may limit the ability of the author to precisely control how an expression is rendered.

Both content and presentation tags are necessary in order to provide the full expressive capability one would expect in a markup language for program code. CODEML offers authors elements for both content and presentation markup. Whether to use one or the other, or a combination of both, depends on what aspects of rendering and interpretation an author wishes to control, and what kinds of re-use he or she wishes to facilitate.

In many common situations, an author or authoring tool may choose to generate either presentation or content markup exclusively. Some applications however are able to make use of both presentation and content information. For these applications it is desirable to provide both forms of markup for the same mathematical expression.

<table>
<thead>
<tr>
<th>Element</th>
<th>Attributes</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Required</td>
<td>Optional</td>
</tr>
<tr>
<td>semantics</td>
<td></td>
<td>(apply</td>
</tr>
<tr>
<td>pcode</td>
<td>format</td>
<td>type, href, id, xlink:*, open, close, separator, indent, breaks</td>
</tr>
<tr>
<td>rawcode</td>
<td>format</td>
<td>type, href</td>
</tr>
</tbody>
</table>

Figure 2.7: The CODEML Interaction Elements
Listing 2.5: Combining markup Styles

```xml
<semantics>
  <ccdef export="find">!−−−−−−see Listing 2.4 −−−−−−</ccdef>
  <pcode format="pseudo">!−−−−−−see Listing 2.3 −−−−−−</pcode>
  <pcode format="java">!−−−−−−see Listing 2.2 −−−−−−</pcode>
  <rawcode format="java">!−−−−−−see Listing 2.1 −−−−−−</rawcode>
</semantics>
```

To combine presentation markup and content markup, CODEML provides the `semantics` element. The first child of this element is a content CODEML expression, and remaining children are `pcode` or `rawcode` elements. The `pcode` elements contain presentation-CODEML representations of the program in the first child, and the `rawcode` elements raw code, i.e. code fragments that have not been marked up in CODEML. The `format` specifies the programming language, the code fragment is in. Note that it is an error to have more than one `pcode` or `rawcode` element with the same `format` attribute. As the example in Listing 2.5 shows, it is allowed to have `pcode` and `rawcode` elements of the same `format`.

In contrast to representation formats like MathML, which allow mixing presentation and content at arbitrary levels, the CODEML format only allows this at the top-level. This is called parallel markup.

### 2.3.1 Parallel Markup by Cross-references

Top-level pairing of independent presentation and content markup is sufficient for many, but not all, situations. Applications that allow treatment of sub-expressions of mathematical objects require the ability to associate presentation, content or information with the parts of an object with mathematical markup. Top-level pairing with a `semantics` element is insufficient in this type of situation; identification of a sub-expression in one branch of `semantics` element gives no indication of the corresponding parts in other branches.

To better accommodate applications that must deal with sub-expressions of large objects, CODEML uses cross-references between the branches of a `semantics` element to identify corresponding sub-structures.

Cross-referencing is achieved using `id` and `xlink:xref` attributes within the branches of a containing `semantics` element. These attributes may optionally be placed on MathML elements of any type.

An `id` attribute and a corresponding `xlink:xref` appearing within the same `semantics` element create a correspondence between sub-expressions.
In general, there will not be a one-to-one correspondence between nodes in parallel branches. For example, a presentation tree may contain elements, such as parentheses, that have no correspondents in the content tree. Therefore CODEML puts the id attributes on the branch with the content tree, which has the finest-grained node structure and the xlink:xref attributes on the presentation-branches.

Let us fortify our intuition about this with an extended development of the find procedure that combines all the material discussed above.

```xml
<?xml version="1.0" encoding="utf-8"?>
<code format="multi" xmlns="http://www.mathweb.org/codeml">
<semantics>
<ccdef export="find" id="find-def">
<ccsym cd="java.dec" name="public-type-function"/>
<apply id="find-type">
<ccsym cd="java.types" name="funtype"/>
<ccsym cd="java.types" name="int"/>
</apply>
<bind>
<ccsym cd="java.proc" name="function"/>
<bvar><ccv name="x"/></bvar>
<apply id="find-body">
<ccsym cd="java.control" name="if"/>
<apply id="cond">
<ccsym cd="java.arith" name="less"/>
<ccv name="s"/>
<ccv name="x"/>
</apply>
<ebb type="number">0</ebb>
</apply>
<apply>
<ccsym cd="java.proc" name="return"/>
<ccv name="x"/>
</apply>
<apply>
<ccsym cd="java.proc" name="return"/>
<apply>
<ccsym cd="union-find" name="find"/>
<apply>
<ccsym cd="java.array" name="select"/>
<ccv name="s"/>
<ccv name="x"/>
</apply>
</apply>
</apply>
</bind>
</ccdef>
</semantics>
</code>
```
<cpg open="" close=""">
</cpg>

<cpg open="" close="" breakO="hard" xref="find--body">
</cpg>

<cpo>
</cpo>

<cpg open="" close="" breakO="hard" xref="find--body">
</cpg>

<cpo>if</cpo>

<cpg open="" close="" breakO="hard" xref="find--body">
</cpg>

<cpo>+</cpo>

<cpg open="" close="" breakO="hard" xref="find--body">
</cpg>

<s[x] is negative</s[x] is negative>

<cpt xref="find--body">
</cpt>

<cpo>
</cpo>

<cpg>
</cpg>

<cpo>then</cpo>

<cpo>return</cpo>

<cpo>x</cpo>

<cpo>
</cpo>

<cpg>
</cpg>

<cpo>otherwise</cpo>

<cpo>return</cpo>

<cpo>s[x]</cpo>

<cpo>
</cpo>

<cpg>
</cpg>

<cpo>way to go!</cpo>

<cpt xref="find--body">
</cpt>

</cpg>
public int find(int x) {
    if (s[x] < 0) return x;
    return find(s[x]);
}
Chapter 3

Conclusion

24 EdNote: say something about what we have done
25 EdNote: say something about why this is useful
Index

&cpt.extra.content;, 5
\&nbsp;, 3

a
  element, 5
abstraction, 16
algorithm, 2
application
  function, 16
apply
  element, 16
  attribute, 15
apply
  element, 16, 20
attribute
  style, 8

background
  attribute
    in \texttt{p-CodeML}, 8, 9
basic data, 16
basic object, 7
bind
  element, 16
  attribute, 15
bind
  element, 16, 17, 20
binding, 16
binding constructor, 16
body, 16
brackets, 9
break, 10
breakC
  attribute
    in \texttt{cpg}, 10
breakO

cd
  attribute
    in ccsym, 14

class
  attribute
    in *, 8

close
  attribute
    in cpg, 9

closing, 9
cmp
  element, 10
code
  coloring, 7
  commented, 2
  country, 11
  highlighting, 7
  program, 2
  raw, 13
code
  element, 20, 22
Code Markup Language, 2
CODEML, i, 2–11, 13–20
color, 7
color
  attribute
    in p-CODEML, 8, 9, 13
comment, 10
commented code, 2
compiler, 2
complex element, 16
computer science, 2
content, 4, 18
content dictionary, 16
Content markup, 17
content markup, 19
constructor
  binding, 16
  country code, 11
cpb
  element, 7
cpbr
  element, 10
cpc
  element, 10
  attribute, 12
cpc
  element, 11, 21, 22
cpd
  element, 10
  attribute, 12
cpd
  element, 11, 21
cpg
  element, 9
  attribute, 6
cpg
  element, 9, 10, 13, 21, 22
cpi
  element, 7
cpo
  element, 7
  attribute, 6
cpo
  element, 7, 21
cpr
  element, 13
cpt
  element, 10
  attribute, 12
cpt
  element, 10, 11, 21
cptype
  element, 7
  attribute, 6
cptype
cstyle
  element, 21
cstyle
  element, 8, 9, 13, 14
data
  basic, 16
data type, 7
de, 11
declaration symbol, 17
defined, 7
defined symbol, 14, 17
definition
  function/procedure, 16
definition schema, 17
descriptive text, 10
development
  environment, 2
  of programs, 2
dictionary
  content, 16
documentation, 2
education, 2
element
  pre(html), 3
  content, 4
  integration, 4
  presentation, 4
element complex, 16
element token, 14
en, 11
exports
  attribute
    in cdef, 17
  extraction, 2
find, 5, 14, 20
find
element
  formal language, 2
  formal parameter, 16
format
  attribute
    in pcode, rawcode, 19
formulae
  mathematical, 3
fr, 11
function
  application, 16
  definition, 16
function call, 16
group
  multilingual, 10
hard, 10
highlighting, 2, 7
HTML, 3, 5, 9
html
  pre, 3
hyper-linking, 2
id
  attribute
    in *, 8
    in c-CodeML, 14, 19, 20
  identifiers, 7
imported, 7
indent
  attribute
    in cpg, 10
  indentation, 2, 9
  integration, 4
ISO 639, 11
JAVA, 5, 14
javascript, 3
language
  formal, 2
languages
  multiple, 10
library, 7
linebreaking, 9
link
  simple, 5
  literate programming, 2
localized, 11
markup
  content, 17, 19
  language
    for mathematical formulae, 3
    for program code, 2
    parallel, 19
    presentation, 17, 19
  mathematical formulae, 3
  Mathematical Markup Language, 3
  mathematical structure, 17
MATHML, i, 3, 4, 19
mathml
  attribute
  multilingual support, 10
  multilingual group, 10
name
  attribute
nl, 11
notational structure, 17
number, 7, 16

object
    basic, 7

\texttt{Obreak}
    attribute
        in \texttt{cpg}, 10

\texttt{OMDoc}, 4

\texttt{open}
    attribute
        in \texttt{cpg}, 9

opening, 9

\texttt{OPENMATH}, i, 3, 14

operator, 7
    built-in, 7
    defined, 7
    imported, 7

parallel markup, 19

parameter
    formal, 16

\texttt{pcode}
    element, 19

\texttt{pcode}
    element, 19

\texttt{pre (html)}, 3

presentation, 4, 18

Presentation markup, 17

presentation markup, 19

pretty-printing, 2

prioritized break, 10

procedure call, 16

procedure definition, 16

program, 2

program code, 2

program development, 2

programmer, 2

programming language, 2

quality assurance, 2

raw code, 13

\texttt{rawcode}
    element, 19

\texttt{rawcode}
    element, 13, 19

\texttt{Sbreak}
    attribute
        in \texttt{cpg}, 10

schema
    definition, 17

scope, 17

\texttt{semantics}
    element, 19

\texttt{semantics}
    element, 19

\texttt{separator}
    attribute
        in \texttt{cpg}, 10

simple link, 5

size
    attribute
        in \texttt{p-CodeML}, 8, 13

space, 9

string, 16

structure
    mathematical, 17
    notational, 17

\texttt{style}
    attribute
        in *, 8

\texttt{style}
    element, 14

style attribute, 8

support
    multilingual, 10

symbol
    declaration, 17
    defined, 14, 17

table, 3

tabstop, 9

text
    descriptive, 10

texts, 2

token element, 14

27
type, 17
data, 7
type
  attribute, 15
    in ccb, 16
    in cpb, 7
    in cpo, 7
type
  element, 20, 21

Unicode, 7

variable, 14

variant
  attribute
    in p-CodeML, 8, 13

XLINK, 5
xlink, 14
xlink:role
  attribute
    in presentation-CodeML, 5
xlink:type
  attribute
    in presentation-CodeML, 5
xlink:xref
  attribute
    in p-CodeML, 19, 20
    in presentation-CodeML, 4, 5

XML, i, 2, 5, 11
xml:lang, 10
  attribute
    in *, 11
    in cpt, 11

XSL, 3
XSLT, 3
## Appendix A

### Quick-Reference Table to the CodeML Elements

<table>
<thead>
<tr>
<th>Element</th>
<th>p.</th>
<th>Type</th>
<th>Required</th>
<th>Optional</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Attribs</td>
</tr>
<tr>
<td>apply</td>
<td>16</td>
<td>Content</td>
<td></td>
<td>id</td>
<td>(apply</td>
</tr>
<tr>
<td>bind</td>
<td>16</td>
<td>Content</td>
<td></td>
<td>id</td>
<td>ccsym, bvar, (apply</td>
</tr>
<tr>
<td>bvar</td>
<td>16</td>
<td>Content</td>
<td></td>
<td>id</td>
<td>ccsym*</td>
</tr>
<tr>
<td>ccdef</td>
<td>16</td>
<td>Content</td>
<td>export</td>
<td>id</td>
<td>ccsym, (apply</td>
</tr>
<tr>
<td>ccsym</td>
<td>16</td>
<td>Content</td>
<td>cd, name</td>
<td>id</td>
<td>EMPTY</td>
</tr>
<tr>
<td>ccv</td>
<td>14</td>
<td>Content</td>
<td>name</td>
<td>id</td>
<td>EMPTY</td>
</tr>
<tr>
<td>cpb</td>
<td>7</td>
<td>PresToken</td>
<td></td>
<td>id, xlink:*</td>
<td>type, variant, style, color, background</td>
</tr>
<tr>
<td>cpbr</td>
<td>10</td>
<td>PresGroup</td>
<td></td>
<td>id, xlink:*</td>
<td>CDATA</td>
</tr>
<tr>
<td>cpc</td>
<td>10</td>
<td>PresText</td>
<td></td>
<td>ctp*</td>
<td></td>
</tr>
<tr>
<td>cpd</td>
<td>10</td>
<td>PresText</td>
<td></td>
<td>ctp*</td>
<td></td>
</tr>
<tr>
<td>cpg</td>
<td>9</td>
<td>PresGroup</td>
<td></td>
<td>id, xlink:*</td>
<td>open, close, separator, indent, breaks</td>
</tr>
<tr>
<td>cpi</td>
<td>7</td>
<td>PresToken</td>
<td></td>
<td>id, xlink:*</td>
<td>variant, style, color, background</td>
</tr>
<tr>
<td>Package</td>
<td>Version</td>
<td>Type</td>
<td>Format</td>
<td>Semantics</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>------</td>
<td>--------</td>
<td>------------</td>
<td></td>
</tr>
<tr>
<td>cpo</td>
<td>7</td>
<td>PresToken</td>
<td>id, xlink:*, type, variant, style, color, background</td>
<td>CDATA</td>
<td></td>
</tr>
<tr>
<td>cpt</td>
<td>10</td>
<td>PresText</td>
<td>variant, style, color, background</td>
<td>CDATA</td>
<td></td>
</tr>
<tr>
<td>cptype</td>
<td>7</td>
<td>PresToken</td>
<td>id, xlink:*, variant, style, color, background</td>
<td>CDATA</td>
<td></td>
</tr>
<tr>
<td>pcode</td>
<td>18</td>
<td>Inter</td>
<td>format</td>
<td>type, href, id, xlink:*, open, close, separator, indent, breaks</td>
<td>(cpg</td>
</tr>
<tr>
<td>rawcode</td>
<td>18</td>
<td>Inter</td>
<td>format</td>
<td>type, href</td>
<td>PCDATA</td>
</tr>
<tr>
<td>semantics</td>
<td>19</td>
<td>Inter</td>
<td></td>
<td>(apply</td>
<td>bind</td>
</tr>
</tbody>
</table>
## Appendix B

### Quick-Reference Table to the CodeML Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>element</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>breaks</td>
<td>cpg</td>
<td>bob-bcb, ob-bcb, bo-bcb, o-bcb, bob-cb, ob-cb, bo-cb, o-cb, bob-bc, ob-bc, bo-bc, o-cb, bob-c, ob-c, bo-c, o-c</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SAY SOMETHING HERE, also about prioritized breaks</td>
</tr>
<tr>
<td>cd</td>
<td>ccsym</td>
<td>theory/module name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name of a theory or a module that exports this symbol</td>
</tr>
<tr>
<td>close</td>
<td>cpg</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The closing bracket of group of CodeML constructs this is inserted into the presentation in place of the opening tag of its cpg element</td>
</tr>
<tr>
<td>color, background</td>
<td>p-CODEML</td>
<td>#rgb,#rgggbb aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, yellow</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The color of the text representation or the background of a presentation-CodeML element</td>
</tr>
<tr>
<td>exports</td>
<td>ccdef</td>
<td>string</td>
</tr>
<tr>
<td></td>
<td></td>
<td>the name of a symbol defined by the ccdef element</td>
</tr>
<tr>
<td>format</td>
<td>pcode, rawcode</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>The format, e.g. the programming language</td>
</tr>
<tr>
<td>id</td>
<td>*</td>
<td>a string that identifies the element</td>
</tr>
<tr>
<td>indent</td>
<td>cpg</td>
<td>number</td>
</tr>
<tr>
<td>name</td>
<td>ccsym</td>
<td>string</td>
</tr>
<tr>
<td>----------</td>
<td>-------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The name the symbol</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>open</th>
<th>cpg</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The opening bracket of group of CodeML constructs; this is inserted into the presentation in place of the opening tag of its cpg element</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>separator</th>
<th>cpg</th>
<th>string</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The separator bracket of group of CodeML constructs this is inserted into the presentation between the children of its cpg element</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size</th>
<th>p-CodeML</th>
<th>small, normal, big, number, v-unit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The font size of the text representation of a presentation-CodeML element</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>cpb, ccb</th>
<th>number</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>the type of the basic object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>type</th>
<th>cpo</th>
<th>built-in, imported, defined</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>the type of the basic object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>variant</th>
<th>p-CodeML</th>
<th>normal, bold, italic, bold-italic, double-struck, bold-fraktur, script, bold-script, fraktur, sans-serif, bold-sans-serif, sans-serif-italic, sans-serif-bold-italic, monospace</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>The font variant of the text of a presentation-CodeML element</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xlink:*</th>
<th>cp*</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>a crossreference to a semantically equivalent content element; the value of xlink:xref (an URIRef) specifies the element, the value of xlink:type must be simple (it is fixed in the DTD), and the xlink:role is a fixed URL to a document that explains this link type.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>xml:lang</th>
<th>cpt</th>
<th>ISO 639: en, de, it, ko, ...</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>the language the text is written in</td>
</tr>
</tbody>
</table>
Appendix C

The CodeML RelaxNG Schema

We reprint the current version of the CodeML RelaxNG schema. The original can be found at https://svn.omdoc.org/repos/codeml/RelaxNG

C.1 The CodeML Schema Driver

```xml
<default namespace = "http://www.mathweb.org/codeml"

equality namespace = "http://www.mathweb.org/codeml/omdoc"

ISO639 = "aa" | "ab" | "af" | "am" | "ar" | "as" | "ay" | "az" | "ba" | "be" | "bg" | "bh" | "bi" | "bn" | "bo" | "br" | "ca" | "co" | "cs" | "cy" | "da" | "de" | "dz" | "el" | "en" | "eo" | "es" | "et" | "eu" | "fa" | "fi" | "fj" | "fo" | "fr" | "fy" | "ga" | "gd" | "gl" | "gn" | "gu" | "ha" | "he" | "hi" | "hr" | "hu" | "hy" | "ia" | "ie" | "ik" | "id" | "is" | "it" | "iu" | "ja" | "jv" | "ka" | "kk" | "km" | "kn" | "ko" | "ku" | "ky" | "la" | "li" | "lo" | "lt" | "lv" | "mg" | "mi" | "mk" | "ml" | "mr" | "mr" | "ms" | "mt" | "my" | "na" | "nc" | "nl" | "no" | "oc" | "om" | "or" | "pa" | "pl" | "ps" | "pt" | "qu" | "rm" | "ru" | "rw" | "sa" | "sd" | "sg" | "sh" | "si" | "sk" | "sl" | "sm" | "si" | "so" | "sq" | "sr" | "ss" | "st" | "su" | "sv" | "sw" | "ta" | "te" | "tg" | "th" | "ti" | "tk" | "tl" | "tn" | "to" | "tr" | "ts" | "tt" | "tw" | "ug" | "uk" | "ur" | "uz" | "vi" | "vo" | "wo" | "xh" | "yi" | "yo" | "za" | "zh" | "zu"
```

C.2 The CodeML Module PRES

Module PRES introduces the presentation CodeML elements.

---

default namespace = "http://www.mathweb.org/codeml"

cp.class = cpg | cpb | cpo | cpi | cpi | cpbr | cptype | cpd | cpc | cpr | cpstyle

cp.common.attrib = common.attrib, attribute xref {xsd:anyURI}?

cp.group.attrib = attribute open { text }?, attribute close { text }?,
attribute separator { text }?,
attribute indent { text }?,
attribute Obreak { text }?,
attribute breakO { text }?,
attribute Cbreak { text }?,
attribute breakC { text }?,
attribute Sbreak { text }?,
attribute breakS { text }?

cp.style.attrib = attribute variant { token }?,
attribute size {text}?,
attribute color { text }?,
attribute background {text}?

cp.token.attrib = cp.common.attrib,cp.style.attrib

cpg = element cpg {cp.class,cp.common.attrib?,cp.group.attrib?}

cpb = element cpb {text,cp.token.attrib?,attribute type {xsd:NCName}}
C.3 The CodeML Module CONT

Module CONT introduces the content CODEML elements.

default namespace = "http://www.mathweb.org/codeml"
cc. class = apply | bind | bvar | ccv | ccb | ccsym
cc.top.class = cc. class | cdef
cc.common.attrib = common.attrib

# application
apply = element apply {cc.class+, cc.common.attrib}

# binding
bind = element bind {ccsym, bvar,cc.class,cc.common.attrib}

# bound variable declarations
bvar = element bvar {ccv+,cc.common.attrib}
C.4 The CodeML Module INTER

Module INTER introduces elements for the interaction of content and presentation CodeML elements. The top-level code element is among them.

```
# variables
ccv = element ccv {attribute name { text }, cc.common.attrib}

# basic language objects
ccb = element ccb {text, attribute type { xsd:NMTOKEN }?, cc.common.attrib?}

# symbols (reserved names)
ccsym = element ccsym {text, attribute cd { text }, attribute name { text }, cc.common.attrib}

# definitions
ccdef = element ccdef {ccsym, cc.class ∗, attribute export { text }, cc.common.attrib}

# symbol declaration
symbol = element symbol {type, cc.common.attrib}

# types
type = element type {cc.class, cc.common.attrib}
```

---

```
C.4 The CodeML Module INTER

Module INTER introduces elements for the interaction of content and presentation CodeML elements. The top-level code element is among them.

```
# A RelaxNG schema for CodeML (presentation and content of program code)
# Interface Module
# Id : omdoc.rnc66342007 − 07 − 1306 : 32 : 51Zkohlhase
# HeadURL : https : //svn.omdoc.org/repos/omdoc/trunk/rnc/omdoc.rnc
# See the documentation and examples at https://www.omdoc.org/codeml
# Copyright © 2008 Michael Kohlhase, Deyan Ginev, Dimitar Misev, Catalin David
# released under the GNU Public License (GPL)

default namespace = "http://www.mathweb.org/codeml"

xcode.attrib = attribute format { text },
  attribute type { text }?,
  attribute href { text }?

semantics = element semantics {cc.top.class, (pcode | rawcode) ∗}

dublincore = grammar (include "MARCRelators.rnc"
  include "dublincore.rnc"
  {dc.date = attribute action {xsd:NMTOKEN}? : attribute who {xsd:anyURI}? , xsd:dateTime
  dc. identifier = attribute scheme {xsd:NMTOKEN}, text
  dc.type = "Dataset" | "Text" | "Collection"
  dc.text = parent common.attrib, parent xml.lang.attrib, text
  dc.person = attribute role {MARCRelators}? , text})

metadata = element metadata {common.attrib, dublincore ∗}

pcode.content = metadata?, cp.class ∗
```
This schema includes the generic schema for Dublin Core Metadata and for the MARC relator set, we include both of them for reference

### C.4.1 The Dublin Core Schema

```xml
# A RelaxNG schema for the Dublin Core elements
Id: omdocdc.rnc
d3992007−05−2515:07:45Zkohlhase
HeadURL: https://svn.omdoc.org/repos/omdoc/branches/omdoc-1.2/rnc/omdocdc.rnc
See the documentation and examples at http://www.omdoc.org
Copyright (c) 2004−2008 Michael Kohlhase, released under the GNU Public License (GPL)

default namespace dc = "http://purl.org/dc/elements/1.1/"

## the various content models, specialize for integration
dc.person = text
dc.publisher = text
dc.text = text
dc.format = text
dc.language = text
dc.rights = text
dc.date = xsd:dateTime
dc.type = text
dc.identifier = text

# the model of the Dublin Metadata initiative (http://purl.org/dc)
start = contributor | creator | rights | subject | title | description | publisher |
| date | type | format | identifier | source | language | relation

contributor = element contributor {dc.person}
creator = element creator {dc.person}
title = element title {dc.text}
subject = element subject {dc.text}
description = element description {dc.text}
publisher = element publisher {dc.publisher}
type = element type {dc.type}
format = element format {dc.format}
language = element language {dc.language}
rel = element relation {dc.relation}
rights = element rights {dc.rights}
date = element date {dc.date}
identifier = element identifier {dc.identifier}
```
C.4.2 The MARC Relators Schema

# the MARC relator set; see http://www.loc.gov/marc/relators
MARCRelators =
"act" | "adp" | "aft" | "ann" | "ant" | "app" | "aqt" |
"arc" | "art" | "ass" | "ass" | "att" | "auc" | "aud" | "auv" |
"aus" | "aut" | "bdd" | "bdj" | "bdk" | "bkp" | "bdn" | "bpd" |
"ccp" | "chr" | "clb" | "cli" | "cll" | "clt" | "cmm" | "cmp" |
"cnd" | "cns" | "coe" | "col" | "com" | "cos" | "cot" | "cov" |
"cpe" | "cph" | "cpl" | "cpt" | "cre" | "crp" | "crr" | "csl" |
"cst" | "ctb" | "cte" | "ctg" | "ctr" | "cts" | "ctt" | "cur" |
"dfd" | "dfe" | "dft" | "dgg" | "dis" | "dlu" | "dnc" | "dmy" |
"dpt" | "dmo" | "drt" | "dsr" | "dst" | "dtc" | "dto" | "dub" |
"egr" | "elt" | "eng" | "etr" | "exp" | "fac" | "fin" | "fmo" |
"frg" | "frq" | "hru" | "hst" | "ill" | "ilu" | "ins" | "inv" |
"ive" | "ivr" | "ltb" | "lee" | "let" | "lre" | "len" | "let" |
"lit" | "lsa" | "lse" | "lsd" | "ltg" | "lyr" | "mdc" | "mod" |
"mrk" | "mte" | "mus" | "nrt" | "opn" | "org" | "orm" | "oth" |
"pat" | "pdb" | "pbt" | "pbr" | "pbt" | "pct" | "pop" | "ppm" |
"prd" | "prf" | "prg" | "prn" | "pro" | "prr" | "pta" | "pte" |
"pth" | "ptt" | "rbr" | "rce" | "rcp" | "red" | "ren" | "res" |
"rpt" | "rpy" | "rse" | "rsp" | "rst" | "rth" | "rmn" | "sad" |
"scl" | "scr" | "sec" | "sgn" | "sgn" | "spk" | "spr" | "spy" |
"stl" | "str" | "str" | "ths" | "tre" | "trt" | "tym" | "tyg" |
"wam" | "wdc" | "wde" | "wit"