The Next 700 Markup Languages

Philip Wadler, Bell Labs

Overview

This talk consists of

- a tribute
- a tutorial
- a half-baked idea of mine
- a better-baked idea of someone else
- an invitation

Peter J. Landin's contributions to computing

- Denotational semantics
  modelled Algol 60 with lambda calculus
- Functional programming
  Iswim, the first functional language
- Abstract machines
  SECD, the first abstract machine
- Continuations
  J, the first jump operator
- Program equivalences

$$\begin{array}{lcl} \textbf{let } x = L\textbf{; } M & \equiv & M \textbf{ where } x = L \\ \textbf{let } y = (\textbf{let } x = L\textbf{; } M)\textbf{; } N & \equiv & \textbf{let } x = L\textbf{; let } y = M\textbf{; } N \end{array}$$

- Abstract syntax
  Physical Iswim, Abstract Iswim, Kernel Iswim
- Domain specific languages
  The Next 700 Programming Languages

The Next 700 Programming Languages

> "... today ... 1,700 special programming languages are used to 'communicate' in 700 application areas." — *Computer Software*, July

A family of unimplemented computing languages is described that is intended to span differences of application area by a unified framework. This framework dictates the rules about the uses of user-coined names, and the conventions about characterizing functional relationships. Within this framework, the design of a language splits into two independent parts. One is the choice of written appearances of programs (or, more generally, their physical representation). The other is the choice of abstract entities (such as numbers, character strings, lists of them, functional relations among them) that can be referred to in the language.

– Peter J. Landin, The Next 700 Programming Languages, *CACM*, March 1966.

Part I

From HTML to XML

XML: an example

```
<html>
  <head>
    <title>Philip Wadler's home page</title>
  </head>
  <body>
    <h1>Philip Wadler's home page</h1>
    <p>My publications include:<ul>
      <li><a href="effect.pdf">The marriage of effects and monads</a>,
        P. Wadler, <em>ICFP 98</em>.</li>
      <li><a href="fj.pdf">Featherweight Java</a>
        A. Igarishi, B. Pierce, and P. Wadler, <em>OOPSLA 99</em>.</li>
    </ul></p>
  </body>
</html>
```

XML: another example

```
<bibliography>
  <paper>
    <author>P. Wadler</author>
    <title>The marriage of effects and monads</title>
    <conference>ICFP 98</conference>
    <href>effects.pdf</href>
  </paper>
  <paper>
    <author>A. Igarishi</author>
    <author>B. Pierce</author>
    <author>P. Wadler</author>
    <title>Featherweight Java: A core calculus for Java</title>
    <conference>OOPSLA 99</conference>
    <href>effects.pdf</href>
  </paper>
</bibliography>
```

XML: Lisp in disguise?

```
(html
  (head
    (title "Philip Wadler's home page"))
  (body
    (h1 "Philip Wadler's home page")
    (p "My publications include:"
      (ul
        (li (a (@href "effects.pdf) "The marriage of effects and monads")
            ", P. Wadler, "
            (em "OOPSLA 98") ".")
        (li (a (@href "fj.pdf") "Featherweight Java")
            ", A. Igarishi, B. Pierce, and P. Wadler,"
            (em "OOPSLA 99") "."))))))
```

DTD: Document Type Description

```
<!ENTITY mix "(h1 | a | p | em | ul | #PCDATA )*">

<!ELEMENT html (head?, body)>

<!ELEMENT head (title?)>

<!ELEMENT title #PCDATA>

<!ELEMENT body %mix;>

<!ELEMENT h1 %mix;>

<!ELEMENT a %mix;>

<!ATTLIST a
  href    CDATA        #REQUIRED>

<!ELEMENT p %mix;>

<!ELEMENT em %mix;>

<!ELEMENT ul ( li )* >

<!ELEMENT li %mix;>
```

DTD: ML in disguise?

```
datatype html     = HTML of head option * body
     and head     = HEAD of title option
     and title    = TITLE of string
     and body     = BODY of mix
     and mixitem  = H1 of mix
                  | A of href * mix
                  | P of mix
                  | EM of mix
                  | UL of li list
                  | PCDATA of string
     and li       = LI of mix
     and href     = HREF of string
withtype mix      = mixitem list
```

Part II

XML Applications and Core Technologies

An ABC of XML applications

- airlines — Air Transport Association of America
- banks — Open Financial Exchange (OFX)
- cell phones — Wireless Markup Language
- astronomy — Astronomical Markup Language
- biology — Bioinformatic Sequence Markup Language
- chemistry — Chemical Markup Language
- IRS and DOD — Extensible Forms Description Language (XFDL)
- graphics — Precision Graphics Markup Language (PGML)
- multimedia — Synchronized Multimedia Integration Language (SMIL)
- standards — DocBook
- standards — W3C Standard DTD
- and finally — Theological ML

WML: Wireless Markup Language

```
<wml>
  <card>
    <p>
      <do type="accept">
        <go href="#card2"/>
      </do>
      Hello world!
      This is the first card...
    </p>
  </card>
  <card id="card2">
    <p>
      This is the second card.  Goodbye.
    </p>
  </card>
</wml>
```

RETS : Real Estate Transaction Standard

```
<PROPERTY>
  <MLNUM>99040102</MLNUM>
  <STATUS>ACT</STATUS>
  <AREA>101</AREA>
  <ADDR>123 Main</ADDR>
  <LISTPR>105000</LISTPR>
</PROPERTY>
<PROPERTY>
  <MLNUM>99030204</MLNUM>
  <STATUS>ACT</STATUS>
  <AREA>101</AREA>
  <ADDR>100 WASHINGTON SQ.</ADDR>
  <LISTPR>550000</LISTPR>
</PROPERTY>
```

Some XML W3C standards

- Namespaces
- DOM — Document Object Model
- RDF — Resource Description Framework
- CSS — Cascading Style Sheets
- XSL — Stylesheet Language (XSL-T, XSL-FO)
- XLL — Linking Language (XLink, XPointer)
- XPath — common subset of XSL and XLL
- XSchema
- XQuery

Namespaces

```
<html:html
  xmlns:html="http://www.w3.org/TR/REC-html40/"
  xmlns:book="http://www.bn.com/Schema/"
  xmlns:person="http://www.cia.gov/Schema/">
  <html:title>Some Great Reads</html:title>
  <book:book>
    <book:title>Conciousness Explained</book:title>
    <book:author>
       <person:title>Prof</person:title> Daniel Dennet
    </book:author>
  </book:book>
</html:html>
```

Modules and Namespaces
both manage name collision, though differently.

Part III

Programming languages for XML

Factorial in PIA

```
<set name='n'>8</set>

<h1>Factorials up to <get name='n'/></h1>

<ul>
  <repeat start='1' stop='&n;' entity='i'>
    <li>factorial(<get name='i'/>) =
      <numeric product>
        <repeat start='1' stop='&i;' entity='k'>
          <get name='k'/>
        </repeat>
      </numeric>
    </li>
  </repeat>
</ul>
```

Factorial in PIA, define a new tag

```
<define element='factorial'>
  <define attribute='j' required/>
  <action>
    <numeric product>
      <repeat start='1' stop='&attributes:j;' entity='k'>
        <get name='k'/>
      </repeat>
    </numeric>
  </action>
</define>
<set name='n'>8</set>
<h1>Factorials up to <get name='n'/></h1>
<ul><repeat start='1' stop='&n;' entity='i'>
  <li>factorial(<get name='i'/>) = <factorial j='&i;'></li>
</repeat></ul>
```

Factorial in XML For All (XFA)

```
<xfa:function Factn>
  <xfa:let n=8>
  <xfa:function Factorial(i)>
     <xfa:if IntEq(i,0)><xfa:val 1/></xfa:if>
     <xfa:else><xfa:val Mult(i,Factorial(Sub(i,1)))/></xfa:else>
  </xfa:function>
  <h2>Factorials up to <xfa:val n></h2>
  <ul>
    <xfa:for i=n>
      <li> factorial(<xfa:val i/>)=<xfa:val Factorial(i)/>
    </xfa:for>
  </ul>
</xfa:function>
```

Factorial in XML For All (XFA), strict form

```
<xfa:function exp="Factn">
  <xfa:let name="n" exp="8">
  <xfa:function exp="Factorial(i)">
     <xfa:if exp="IntEq(i,0)"><xfa:val exp="1"/></xfa:if>
     <xfa:else><xfa:val exp="Mult(i,Factorial(Sub(i,1)))"/></xfa:else>
  </xfa:function>
  <h2>Factorials up to <xfa:val exp="n"></h2>
  <ul>
    <xfa:for name="i" exp="n">
      <li> factorial(<xfa:val exp="i"/>)=<xfa:val exp="Factorial(i)"/>
    </xfa:for>
  </ul>
</xfa:function>
```

Part IV

X

Factorial in X

```
factorial(k) = if(k == 0, 1, k*factorial(k-1)),

n = 8,

{ <h1>Factorials up to {n}</h1>

  <ul>

    { for(i = 0, i <= n, i = i+1,

        {<li> factorial({i}) = {factorial(i)} </li>}

      ) }

  </ul> }
```

The X isomorphism

$\{t(e_1,\ldots,e_n)\} \equiv$ <t> $\{e_1\}$ ... $\{e_n\}$ </t>

$\{t()\} \equiv$ <t/>

$\{t(@a_1("s_1"),\ldots,@a_m("s_m"),e_1,\ldots,e_n)\}$
$\equiv$ <t a_1="s_1"...a_m="s_m">$\{e_1\}$...$\{e_n\}$</t>

$\{e_1 + \cdots + e_n\} \equiv$ <plus-sign> $\{e_1\}$ ... $\{e_n\}$ </plus-sign>

$\{n\} \equiv$ <var name="n"/>, where n is a name

$\{i\} \equiv$ <int val="i"/>, where i is an integer

$\{"s"\} \equiv$ <str val="s"/>, where s is a string

$\{\{x\}\} \equiv x$

Factorial in X, strict form

```
<equals-sign>
  <factorial><var name="k"/></factorial>
  <if>
    <equals-sign-equals-sign>
       <var name="k"/>
       <int val="0"/>
    </equals-sign-equals-sign>
    <int val="1"/>
    <asterisk-sign>
      <var name="k"/>
      <factorial>
        <hyphen-minus-sign>
          <var name="k"/>
          <int val="1"/>
        </hyphen-minus-sign>
      </factorial>
    </asterisk-sign>
  </if>
</equals-sign>
<equals-sign><var name="n"><int val="8"></equals-sign>
```

```
<h1>Factorials up to <var name="n"></h1>
<ul>
  <for>
    <equals-sign><var name="i"/><int val="0"/></equals-sign>
    <less-than-sign-equals-sign>
       <var name="i"/>
       <var name="n"/>
    </less-than-sign-equals-sign>
    <equals-sign>
      <var name="i">
      <plus-sign><var name="i"/><int val="1"/></plus-sign>
    </equals-sign>
    <li>
      factorial(<var name="i"/>) = <factorial><var name="i"/></factorial>
    </li>
  </for>
</ul>
```

Factorial in X, lazy form

```
factorial(k) = if(k == 0, 1, k*factorial(k-1)),

n = 8,

h1({Factorials up to {n}})

ul(

  for(i = 0, i <= n, i = i+1,

    li({factorial({i}) = {factorial(i)}})

  )

)
```

Part V

XSLT

XSLT: an example

```
<xsl:template match="page">
  <html>
    <head>
      <title><xsl:value-of select="title"/></title>
    </head>
    <body>
      <h1><xsl:value-of select="title"/></h1>
      <ul><xsl:apply-templates select="bib/paper"/></ul>
    </body>
  </html>
</xsl:template>
```

XSLT: an example, continued

```
<xsl:template match="paper">
  <li>
    <xsl:apply-templates select="title">
      <xsl:with-param name="href" select="href"/>
    </xsl:apply-templates>
    <xsl:text>, </xsl:text>
    <xsl:apply-templates select="author"/>
    <xsl:text>, </xsl:text>
    <xsl:apply-templates select="conference"/>
    <xsl:text>.</xsl:text>
  </li>
</xsl:template>
```

XSLT: an example, continued

```
<xsl:template match="paper/title">
  <xsl:param name="href">
  <a href=$href><xsl:apply-templates/></a>
</xsl:template>
<xsl:template match="paper/conference">
  <em><xsl:applytemplates/></em>
</xsl:template>
```

XSLT: an example, continued

```
<xsl:template match="paper/author[position()=1]">
  <xsl:apply-templates>
</xsl:template>
<xsl:template match="paper/author[position()=last()]">
  <xsl:text>, and <xsl:text><xsl:apply-templates>
</xsl:template>
<xsl:template match="paper/author">
  <xsl:text>, <xsl:text><xsl:apply-templates>
</xsl:template>
```

Part VI

Xduce

Haruo Hosoya, Benjamin C. Pierce,

and Peter Buneman

University of Pennsylvania

Xduce: an example

```
type Html          = html [Head?, Body]

type Head          = head [Title]

type Title         = title [String]

type Body          = body [Mix]

type Mixitem       = h1 [Mix]
                   | a [Href, Mix]
                   | p [Mix]
                   | em [Mix]
                   | ul [Li*]
                   | String

type Li            = li [Mix]

type Href          = href [String]

type Mix           = Mixitem*
```

Xduce: an example, continued

```
type Paper        = paper[Author+,Title,Conference,Href]
type Author       = author[String]
type Title        = title[String]
type Conference   = conference[String]
type Href         = href[String]
fun do_page : Html =
  page[title[t : String], bib[b: Paper*]] ->
    html[
      head[title[t]],
      body[h1[t], ul[do_bib(b)]]]
fun do_bib : Li* =
  p : Paper, b : Paper* ->
   do_paper(p), do_bib(b)
| () ->
   ()
```

Xduce: an example, continued

```
fun do_paper : Li =
  paper[a : Author+, t : Title, c : Conference, h : Href] ->
    li[do_title(h,t), ", ", do_authors(a), ", ", do_conference(c), "."]
fun do_title : Mix =
  href[h : String], title[t : String] ->
    a[href[h],t]
fun do_conference : Mix =
  conference[c] ->
    em[c]
```

Xduce: an example, continued

```
fun do_authors : Mix =
  a : Author ->
    do_author(a)
| a : Author, b : Author ->
    do_author(a), " and ", do_authors(b)
| a : Author, b : (Author,Author+) ->
    do_author(a), ", ", do_authors(b)
fun do_author : Mix =
  author[a : String] -> a
```

To do list

Add the following to the syntax of X
and the type system of Xduce:

- higher-order functions and polymorphism
- pattern matching and queries
- namespaces and modularity

And then onto the next 700 markup languages!

An invitation

I hope to have convinced you that:

- XML raises many interesting problems
- programming language experts know about these problems
- a good solution to these problems can make you famous

I invite you to contribute to the future of XML! They need you!

For more on XML see:

http://www.cs.bell-labs.com/~wadler/xml/