Web programming without tiers

Philip Wadler
University of Edinburgh
wadler@inf.ed.ac.uk
Team Links

Jeremy Yallop, Rodney Topor, Ezra Cooper
Coauthors

Cooper, Topor & Yallop
Buneman & Wong
Hejlsberg & Meijer
Hasoya & Pierce
Boag, Chamberlin, Fernández, Florescue, Robie, & Siméon
Graunke, Findler, Krishnamurthi & Felleisen
Quenniec
Graham
Armstrong, Virding, Williams, & Wikström
Eich
A Grand Challenge

Design a programming language with a sound basis in theory that becomes the leader in its domain.
A *Jolly Good* Challenge

Design a programming language with a sound basis in theory that becomes the leader in its domain.
Wadler’s theorem of language adoption

A programming language will be adopted if and only if it permits its users to do something that cannot be done in any other way.
Wadler’s theorem of language adoption

A programming language will be adopted if and only if it permits its users to

*boldly go where no programming language has gone before.*
Three-tier model

Browser
(HTML, XML, Javascript, Flash)

form \(\uparrow\) \(\downarrow\) result

Server
(Java, Python, Perl, PHP)

query \(\downarrow\) \(\uparrow\) result

Database
(SQL, XQuery)
Links builds on successes of functional programming

- **Databases** Kleisli, LINQ
  Compile Links into SQL, XQuery

- **XML** Xduce, XQuery
  XML support with regular expression types

- **Continuations** PLT Scheme, Yahoo stores
  Continuations for web dialogue

- **Distribution** Erlang, JoCaml
  Reliability as in Erlang/OTP

- **Javascript** AJAX
  Compile Links into Javascript
Comprehensions

\[
\begin{align*}
\left\{ (x, y) \mid x &\leftarrow [1, 2, 3], y \leftarrow \{\text{'a'}, \text{'b'}\} \right\} \\
&= \text{join} \left[ \left\{ (x, y) \mid y \leftarrow \{\text{'a'}, \text{'b'}\} \right\} \mid x \leftarrow [1, 2, 3] \right]\right] \\
&= \text{join} \left[ \left\{ (x, \text{'a'}), (x, \text{'b'}) \right\} \mid x \leftarrow [1, 2, 3] \right]\right] \\
&= \text{join} \left[ \left\{ (1, \text{'a'}), (1, \text{'b'}) \right\}, \left\{ (2, \text{'a'}), (2, \text{'b'}) \right\}, \left\{ (3, \text{'a'}), (3, \text{'b'}) \right\} \right]\right] \\
&= \left\{ (1, \text{'a'}), (1, \text{'b'}), (2, \text{'a'}), (2, \text{'b'}), (3, \text{'a'}), (3, \text{'b'}) \right\}
\end{align*}
\]
Monads and Comprehensions

(1) \[ [ t | () ] = \text{unit } t \]

(2) \[ [ t | x \leftarrow u ] = \text{map } (\lambda x. t) u \]

(3) \[ [ t | (p, q) ] = \text{join } [ [ t | q ] | p ] \]

(1\textquotesingle) \[ \text{unit } x = [ x ] \]

(2\textquotesingle) \[ \text{map } f xs = [ f x | x \leftarrow xs ] \]

(3\textquotesingle) \[ \text{join } xss = [ x | xs \leftarrow xss, x \leftarrow xs ] \]
Monad laws and Comprehension laws

(I) \( \text{join} \cdot \text{unit} = \text{id} \)

(II) \( \text{join} \cdot \text{map unit} = \text{id} \)

(III) \( \text{join} \cdot \text{join} = \text{join} \cdot \text{map join} \)

(I') \[ t \mid () , q \] = \[ t \mid q \]

(II') \[ t \mid q , () \] = \[ t \mid q \]

(III') \[ t \mid (p , q) , r \] = \[ t \mid p , (q , r) \]
# Relational Data

## BOOKS

<table>
<thead>
<tr>
<th>title</th>
<th>isbn</th>
<th>year</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Can You Do With a Shoe?</td>
<td>0613733266</td>
<td>1997</td>
</tr>
<tr>
<td>Where the Wild Things Are</td>
<td>0060254920</td>
<td>1963</td>
</tr>
</tbody>
</table>

## AUTHORS

<table>
<thead>
<tr>
<th>author</th>
<th>isbn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beatrice Schenk de Regniers</td>
<td>0613733266</td>
</tr>
<tr>
<td>Maurice Sendak</td>
<td>0613733266</td>
</tr>
<tr>
<td>Maurice Sendak</td>
<td>0060254920</td>
</tr>
</tbody>
</table>
Relational Query

**SQL**

```sql
select b.title, a.author
from BOOKS b, AUTHORS a
where b.isbn = a.isbn
    and b.year < 2000
```

**Kleisli (Buneman & Wong & others)**

```csharp
{ (title: b.title, author: a.author) |
  \b <--- BOOKS, \a <--- AUTHORS,
  b.isbn = a.isbn, b.year < 2000 }
```

**LINQ (Heijlsberg & Meijer & others)**

```csharp
from b in BOOKS
from a in AUTHORS
where b.isbn == a.isbn && b.year < 2000
select new { title=b.title, author=a.author };
```
An odd relational Query

Kleisli

\[
\{ (\text{title}: \ b.\text{title}, \ \text{author}: \ a.\text{author}) \mid \\
\quad \text{b} \leftarrow \text{BOOKS}, \ \text{a} \leftarrow \text{AUTHORS}, \\
\quad \text{b}.\text{isbn} = \text{b}.\text{isbn}, \ \text{b}.\text{year} < 2000, \ \text{odd}(\text{b}.\text{year}) \} 
\]

Optimized Kleisli

\[
\{ (\text{title}: \ t, \ \text{author}: \ a) \mid \\
\quad (\text{title}: \ t, \ \text{year}: \ y, \ \text{author}: \ a) \leftarrow \{ (\text{title}: \ b.\text{title}, \ \text{author}: \ a.\text{author}, \ \text{year}: \ b.\text{year}) \mid \\
\quad \text{t} \leftarrow \text{BOOKS}, \ \text{a} \leftarrow \text{AUTHORS}, \\
\quad \text{t}.\text{isbn} = \text{a}.\text{isbn}, \ \text{t}.\text{year} < 2000 \} \}
\quad \text{odd}(y) \} 
\]
Kleisli for bioinformatics

localblast-blastp (#name: "scop-blast", #db: "scopseq");
localblast-blastp (#name: "pat-blast", #db: "patseq");
scop-add "scop";
setindex-access (#name: "sid2seq", #file: "scopseq",
    #key: "#sid");

{ (#sf: (#desc: xinfo.#desc.#sf, #hit: x.#accession,
            #pscore: x.#pscore),
    #bridge: (#hit: s, #patent: p.#title, #pscore: p.#pscore))
| \x <- process SEQ using scop-blast, x.#pscore <= PSCORE,
  \xinfo <- process <#sidinfo: x.#accession> using scop,
  \s <- process <#numsid: xinfo.#type.#sf> using scop,
  \y <- process <#key: s> using sid2seq,
  \p <- process y.#seq using pat-blast, p.#pscore <= PSCORE };

Kleisli was first to perform “twelve impossible queries” identified by DoE
Workshop for Human Genome Project
<books>
    <book>
        <title>Where the Wild Things Are</title>
        <author>Maurice Sendak</author>
        <isbn>0060254920</isbn>
        <year>1963</year>
    </book>
    <book>
        <title>What Can You Do With a Shoe?</title>
        <author>Beatrice Schenk de Regniers</author>
        <author>Maurice Sendak</author>
        <isbn>0613733266</isbn>
        <year>1997</year>
    </book>
</books>
XML Query

XQuery

for $b from input()//books/book
  $a from $b/author
where $b/year < 2000
return
  <book>{ $b/title, $a }</book>

Kleisli

{ (title: b.title, author: a) |
  \b <- BOOKS, \a <- t.authors,
  b.year < 2000 }

LINQ

from b in BOOKS
from a in b.authors
where b.isbn == a.isbn && b.year < 2000
select (b.title, a.author)
Links

- General-purpose, compiles to SQL or XQuery
  differs for Kleisli

- No syntactic distinction, one expression may query multiple sources
  differs from LINQ

- Other related work
  Mnesia in Erlang (Mattsson, Nilsson & Wikstrom)
  Natural Expert (Hutchison, Neuhaus, Schmidt-Schauss & Hall)
Part II

Xduce:
Regular expression types for XML
XML data

<addrbook>
  <person>
    <name>Haruo Hosoya</name>
    <email>hahosoya@kyoto-u</email>
    <email>hahosoya@upenn</email>
  </person>
  <person>
    <name>Benjamin Pierce</name>
    <email>bcpierce@upenn</email>
    <tel>123-456-789</tel>
  </person>
</addrbook>

Hosoya and Pierce
Xduce types

```haskell
type Addrbook = addrbook[Person*]
type Person = person[Name, Email*, Tel?]
type Name = name[String]
type Email = email[String]
type Tel = tel[String]

type TelBook = telbook[TelPerson*]
type TelPerson = person[Name, Tel]
```
XML Schema

<xs:element name="addrbook">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Person"
        minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="addrbook">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="email" type="xs:string"/>
        minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="tel" type="xs:string"/>
        minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
Xduce transformation

```haskell
fun telbook (doc : Addrbook) : TelBook = 
  match doc with
  addrbook[val persons as Person*] ->
    telbook[telpersons(persons)]

fun telpersons (val ps as Person*) : TelPerson* = 
  match ps with
  person[name[val n as String], Email*,
    tel[val t as String]],
    val rest as Person*
      -> person[name[n], tel[t]],
        telpersons(rest)
  | person[name[val n as String], Email*],
    val rest as Person*
      -> telpersons(rest)
  | ()
    -> ()
```
XQuery transformation

<telbook>{
    for $person in input()/addrbook/person[tel] return
    <person>{$person/name, $person/tel}</person>
}</telbook>
XQuery 1.0: An XML Query Language

W3C Working Draft 16 August 2002

This version:
http://www.w3.org/TR/2002/WD-xquery-20020816/

Latest version:
http://www.w3.org/TR/xquery/

Previous versions:
http://www.w3.org/TR/2002/WD-xquery-20020430/
http://www.w3.org/TR/2001/WD-xquery-20011220/
http://www.w3.org/TR/2001/WD-xquery-20010607/

Editors:
Scott Boag (XSL WG), IBM Research <scott_boag@us.ibm.com>
Don Chamberlin (XML Query WG), IBM Almaden Research Center <chamberlin@almaden.ibm.com>
Mary F. Fernandez (XML Query WG), AT&T Labs <mff@research.att.com>
Daniela Florescu (XML Query WG), XQRL <dana@xqrl.com>
Jonathan Robie (XML Query WG), DataDirect Technologies <jonathan.robie@datadirect-technologies.com>
Jérome Siméon (XML Query WG), Bell Labs, Lucent Technologies <simeon@research.bell-labs.com>

Copyright © 2002 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use, and software licensing rules apply.
XQuery 1.0 and XPath 2.0 Formal Semantics

W3C Working Draft 16 August 2002

This version:
http://www.w3.org/TR/2002/WD-query-semantics-20020816/

Latest version:
http://www.w3.org/TR/query-semantics/

Previous versions:
http://www.w3.org/TR/2002/WD-query-semantics-20020326/
http://www.w3.org/TR/2001/WD-query-semantics-20010607/
http://www.w3.org/TR/2001/WD-query-algebra-20010215/
http://www.w3.org/TR/2000/WD-query-algebra-20001204/

Editors:
Denise Draper (XML Query WG), Nimble Technology <ddraper@nimble.com>
Peter Fankhauser (XML Query WG), Infonyte GmbH <fankhaus@infonyte.com>
Mary Fernández (XML Query WG), AT&T Labs - Research <mff@research.att.com>
Ashok Malhotra (XML Query and XSL WGs), Microsoft <ashokma@microsoft.com>
Kristoffer Rose (XSL WG), IBM Research <krisrose@us.ibm.com>
Michael Rys (XML Query WG), Microsoft <mrys@microsoft.com>
Jérôme Siméon (XML Query WG), Lucent Technologies <simeon@research.bell-labs.com>
Philip Wadler (XML Query WG), Avaya <wadler@avaya.com>

Copyright © 2002 W3C® (MIT, INRIA, Keio), All Rights Reserved. W3C liability, trademark, document use, and software licensing rules apply.
3.3.2 Matches

Notation

The judgment

\[ \text{Value matches Type} \]

holds when the given value matches the given type.

Semantics

This judgment is specified by the following rules.

The empty sequence matches the empty sequence type.

\[
\text{statEnv} \vdash \emptyset \text{ matches } \emptyset
\]

If two values match two types, then their sequence matches the corresponding sequence type.

\[
\begin{align*}
\text{statEnv} \vdash Value_1 \text{ matches Type}_1 \\
\text{statEnv} \vdash Value_2 \text{ matches Type}_2
\end{align*}
\]

\[
\text{statEnv} \vdash Value_1, Value_2 \text{ matches Type}_1, Type_2
\]
Links

• Also use regular expressions for lists
  A, A?, A+, A*
  A? corresponds to maybe type of Haskell

• XML syntax that works for cut-and-paste
  \(<\text{greet}>\text{Hello, } \{x\}\</\text{greet}> \text{ vs. } \langle\text{greet}>'\text{Hello, } ', x\langle/\rangle\)

• Other related work
  Xtatic (Gapayev, Levin & Pierce)
  Cduce (Benzaken, Castagna & Frisch)
  Jwig (Schwartbach & Møller)
Part III

PLT Scheme:
Continuations for the Web
Orbitz

[Image of Orbitz website]

Graunke, Findler, Krishnamurthi, Felleisen
slides courtesy of Shriram Krishnamurthi
Orbitz
Orbitz
Orbitz
Orbitz

Hotel & room details

Best Western Sweetgrass Inn
1540 Savannah Highway
Charleston, SC 29407

check in Wed, Nov 20, 2002
check out Fri, Nov 22, 2002

guests 1

room Bwi best rate *our best v

Hotel & room details

Residence Inn by Marriott Charleston Downtown Riverview
90 Ripley Point Drive
Charleston, SC 29407

check in Wed, Nov 20, 2002
check out Fri, Nov 22, 2002

guests 1

room American auto assoc + st
(bath) * id required at ch
Orbitz
Best Western Sweetgrass Inn
1540 Savannah Highway
Charleston, SC 29407

Room cost summary

Residence Inn by Marriott Charleston
Downtown Riverview
90 Ripley Point Drive
Charleston, SC 29407

Hotel & room details
Orbitz

Est. total cost $214.20*

Residence Inn by Marriott Charleston Downtown Riverview
90 Ripley Point Drive
Charleston, SC 29407

check-in Wed, Nov 20, 2002
check-out Fri, Nov 22, 2002
guests 1
number of nights 2

Shriram Krishnamurthi

*phone number
401-453-5890

optional special request
no special request

Frequent guest information
no frequent guest program
stored frequent guest program

Room cost summary
<table>
<thead>
<tr>
<th>Residence Inn by Marriott</th>
<th>Shriram Krishnamurthi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Charleston Downtown Riverview</td>
<td>check-in</td>
</tr>
<tr>
<td>90 Ripley Point Drive</td>
<td>check-out</td>
</tr>
<tr>
<td>Charleston, SC 29407</td>
<td>guests</td>
</tr>
<tr>
<td>number of nights</td>
<td>2</td>
</tr>
</tbody>
</table>

- Est. total cost: $214.20
- Phone number: 401-453-5890
- Special request: no special request
Continuations

The Influence of Browsers on Evaluators or, Continuations to Program Web Servers

Christian Queinnec
Université Paris 6 — Pierre et Marie Curie
LIP6, 4 place Jussieu, 75232 Paris Cedex — France
Christian.Queinnec@lip6.fr

http://my.scheme/eval/*-2-(read)

But then our hero hits the Back button of the browser

Proposition 1:
A browser is a device that can invoke continuations multiply.

Christian Queinnec (ICFP 2000)
Continuations

http://my.scheme/

What to eval?

tick-a-tick-a-tick...
Continuations

http://my.scheme/eval?exp=(*+2+(read))

What to eval?

(* 2 (read))

tick-a-tick-a-tick... click!

What to read?
Continuations

http://my.scheme/resume?continuation=k87&exp=3

**What to read?**

```plaintext
3
```

**tick! click!**

**Result is**

```plaintext
6
```
But then our hero hits the Back button of the browser
Continuations

http://my.scheme/resume?continuation=k87&exp=4

What to read?

Result is

4
tick! tick! click!

also used by Paul Graham for Yahoo stores
Links

- What are the right scopes? session scope vs. global scope key question for web applications!
- “Scalable” state stored in client, not in server
- Mechanisms hooks to choose implementation technique URL parameters vs. hidden fields cryptographically protect state kept in client
- Other related work Mawl (Ramming, Atkins, Ball, Bruns, Cox) WASH (Thiemann)
Part IV

Erlang:

Communication via values
Erlang: An area server

```erlang
start() ->
    register(area_server,
        spawn(fun() -> loop(0) end)).
loop(Tot) ->
    receive
        {Pid, {square, X}} ->
            Pid ! X*X,
            loop(Tot + X*X);
        {Pid, {rectangle, [X,Y]}} ->
            Pid ! X*Y,
            loop(Tot + X*Y);
        {Pid, areas} ->
            Pid ! Tot,
            loop(Tot)
    end.
```

Armstrong, Virding, Williams & Wikström
Erlang: Generic server

```
start(Name, Data, Fun) ->
    register(Name,
        spawn(fun() -> loop(Data, Fun) end)).
Rpc(Name, Query) ->
    Tag = ref(),
    Name ! {query, self(), Tag, Query},
    receive
        {Tag, Reply} -> Reply
    end.
loop(Data, Fun) ->
    receive
        {query, Pid, Tag, Query} ->
            {Reply, Data1} = Fun(Query, Data),
            Pid ! {Tag, Reply},
            loop(Data1, Fun)
    end.
```
Erlang: Instantiating the Generic Server

start() ->
    start(area_server, 0, handler/2).
handler({square, X}, Tot) ->
    {X*X, Tot + X*X};
handler({rectangle, [X,Y]}, Tot) ->
    {X*Y, Tot + X*Y};
handler(areas, Tot) ->
    {Tot, Tot}.
Erlang: Instantiating a Replicated Server

```erlang
start() ->
    start_replicated(area_server, 0, handler/2).
handler({square, X}, Tot) ->
    {X*X, Tot + X*X};
handler({rectangle, [X,Y]}, Tot) ->
    {X*Y, Tot + X*Y};
handler(areas, Tot) ->
    {Tot, Tot}.
```
Links

- Regular expressions for session types
  ```plaintext
  session AreaServer =
  (square[Int] | rectangle[Int,Int] | areas[])*
  session InitializedAreaServer =
  initialize[],
  (square[Int] | rectangle[Int,Int] | areas[])*
  ```

- User interface as a process
  ```plaintext
  session DOM =
  (root[HTML] | todolist[li[String]*])*  
  ```

- Resumption-passing style
  Server processes hibernate on client

- Other related work
  JoCaml (Fournet & Gonthier)
  Timber (Carlsson, Nordlander & Kieburtz)
Part V

Javascript:
The world’s most-widely deployed functional language
Javascript is a functional language

Array.prototype.reduce=function(templateFunction) {
    var l=this.length;
    var s='';
    for (var i=0;i<l;i++) s+=templateFunction(this[i]);
    return s;
}

function wrap(tag) {
    var stag='<'+tag+'>';
    var etag='</'+tag.replace(/s. * /,'')+'>';
    return function(x) {
        return stag+x+etag;
    }
}

document.write(
    '<TABLE><TR>'+
    arr.reduce(wrap('TD class="small"'))+
    '</TR></TABLE>'
);
AJAX
Part VI

Java:

Another widely deployed functional language
Java generics
Part VII

Links
Hope and Links
Hope and Links

Links
(Bruntsfield Links)
Wadler et al (2005)

Hope
(Hope Park Square)
Burstall, MacQueen, Sannella (1980)
Links meeting, 6 April 2005
Links meeting, 6 April 2005
Part VIII

A Links program state in client
To do list

Buy groceries  done
Deliver lecture  done
Vote  done

add
main() { todo([]) }
todo(items) {
  <html><body>
  <h1>Items to do</h1>
  <table>
    <tr>
      <td>{item}</td>
      <td>
        <form l:action="{todo(items\[[item\])}]">
          <input type="submit" value="done"/>
        </form>
      </td>
    </tr>
  </table>
  <form l:action="{todo(items++\[new\])}">
    <input l:name="{new}" type="text" size="40">
    <input type="submit" value="add"/>
  </form>
  </body></html>
}
Part IX

A Links program
state in server
table TODO of (name : String, item: String)
lookup(n) { [ i | (name:n,item:i) <- TODO ] }
add(n,i) {
    insert into TODO values (name:n, item:i);
    todo(name)
}
remove(n,i) {
    remove from TODO values (name:n, item:i);
    todo(name)
}
main() {
<html><body>
    <h1>Login</h1>
    <form l:action="todo(name)">
        <input l:name="{name}" type="text" size="40">
        <input type="submit" value="login"/>
    </form>
</body></html>
todo(name) {
    let items = lookup(name) in
    <html><body>
    <h1>Items to do</h1>
    <table>
        for item in items return
        <tr>
            <td>{item}</td>
            <td>
                <form l:action="{remove(name, item)}">
                    <input type="submit" value="done"/>
                </form>
            </td>
        </tr>
    </table>
    <form l:action="{add(name, new)}">
        <input l:name="{new}" type="text" size="40">
        <input type="submit" value="add"/>
    </form>
    </body></html>
}
Part X

Conclusions
Other ideas

- **Multimethods**
  Integrate functional and OO styles

- **Type classes**
  Semantics should determine types, not types determine semantics

- **Lists and dictionaries as data structures**
  Regular expression matching for lists — down with cons!

- **Testing and validity** Contracts, Quickcheck
Antinomies (technical)

• Pure or effectful?
  Effects with effect types

• Lazy vs. strict?
  Strict with support for lazy closures

• Type inference vs. subtyping?
  Give up on type inference

• How to import imperative libraries?
  Hide them in a process
Antinomies (social)

- **Is it really research?**
  Hard for academics to build real systems

- **Is it too much research?**
  Haskell and SML built on strong consensus

- **How do we build a community?**
  We can only succeed if we do it together
</Links>