Links

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

Less than a Grand 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 progamming language has gone before.

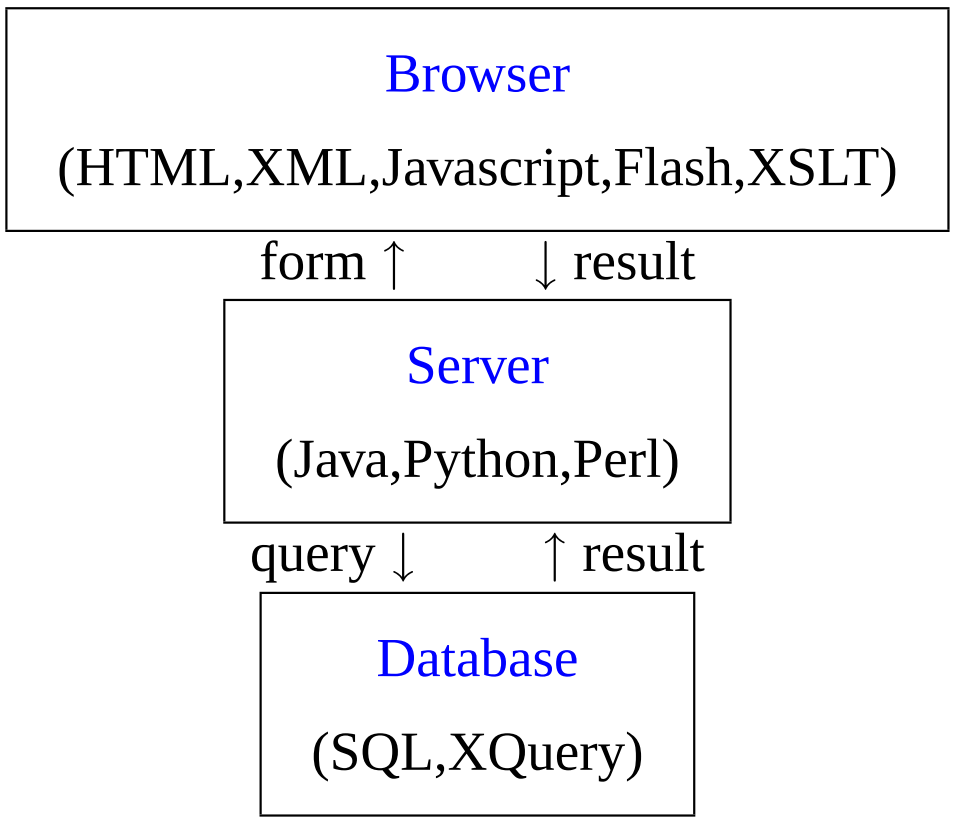
Success stories

- Klesli (databases)
- XDuce and XQuery (XML, databases)
- PLT Scheme (web applications)
- Erlang (distribution)

Why can’t you do this in Haskell or ML?

- Kleisli (SQL compilation, record and variant types)
- Xduce (Regular expression types)
- PLT Scheme (Variable binding, session types)
- Erlang (Distributed communication, channel types)

Three-tier model



Kleisli

Comprehensions for Queries

Comprehensions

$$[\ (x, y) \mid x \leftarrow [1, 2, 3],\ y \leftarrow [\text{‘a’}, \text{‘b’}]\ ]$$

$$=$$

$$join\ [\ [\ (x, y) \mid y \leftarrow [\text{‘a’}, \text{‘b’}]\ ]\ \mid x \leftarrow [1, 2, 3]\ ]$$

$$=$$

$$join\ [\ [\ (x, \text{‘a’}), (x, \text{‘b’})\ ]\ \mid x \leftarrow [1, 2, 3]\ ]$$

$$=$$

$$join\ [\ [(1, \text{‘a’}), (1, \text{‘b’})],\ [(2, \text{‘a’}), (2, \text{‘b’})],\ [(3, \text{‘a’}), (3, \text{‘b’})]\ ]$$

$$=$$

$$[\ (1, \text{‘a’}), (1, \text{‘b’}), (2, \text{‘a’}), (2, \text{‘b’}), (3, \text{‘a’}), (3, \text{‘b’})\ ]$$

Monads and Comprehensions

$$
\begin{array}{lrcl}
(1) & [\, t \mid () \,] & = & unit\, t \\
(2) & [\, t \mid x \leftarrow u \,] & = & map\, (\lambda x.\, t)\, u \\
(3) & [\, t \mid (p, q) \,] & = & join\, [\, [\, t \mid q \,] \mid p \,] \\
 & & & \\
(1') & unit\, x & = & [\, x \,] \\
(2') & map\, f\, xs & = & [\, f\, x \mid x \leftarrow xs \,] \\
(3') & join\, xss & = & [\, x \mid xs \leftarrow xss,\, x \leftarrow xs \,]
\end{array}
$$

Monad laws and Comprehension laws

$$
\begin{array}{lll}
(I) & join \cdot unit & = & id \\
(II) & join \cdot map\, unit & = & id \\
(III) & join \cdot join & = & join \cdot map\, join \\
 & & & \\
(I') & [\, t \mid (),\, q\,] & = & [\, t \mid q\,] \\
(II') & [\, t \mid q,\, ()\,] & = & [\, t \mid q\,] \\
(III') & [\, t \mid (p,\, q),\, r\,] & = & [\, t \mid p,\, (q,\, r)\,]
\end{array}
$$

Comprehension laws

$$
\begin{array}{lllll}
(a) & [\, x \mid x \leftarrow u \,] & = & u \\
(b) & [\, t \mid p,\ x \leftarrow [u|q],\ r \,] & = & [\, t[u/x] \mid p,\ q,\ r[u/x] \,]
\end{array}
$$

Relational Data

TITLES

title	isbn	year
What Can You Do With a Shoe?	0613733266	1997
Where the Wild Things Are	0060254920	1963

AUTHORS

author	isbn
Beatrice Schenk de Regniers	0613733266
Maurice Sendak	0613733266
Maurice Sendak	0060254920

Relational Query

SQL

```
select t.title, a.author
from TITLES t, AUTHORS a
where t.isbn = a.isbn
  and t.year < 2000
```

Kleisli

```
TITLES : {(title: String, isbn: Integer, year: Date)}
AUTHORS : {(author: String, isbn: Integer)}

{ (title: t.title, author: a.string) |
  \t <--- TITLES, \a <--- AUTHORS,
  t.isbn = a.isbn, t.year < 2000 }
```

An odd relational Query

Kleisli

```
{ (title: t.title, author: a.string) |
  \t <--- TITLES, \a <--- AUTHORS,
  t.isbn = a.isbn, t.year < 2000, odd(t.year) }
```

Optimized Kleisli

```
{ (title: t, author: a) |
   (title: \t, year: \y, author: \a)
    <- process("select t.title, a.author
                from TITLES t, AUTHORS a
                where t.isbn = a.isbn
                and t.year < 2000"),
    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

XML Data

```
<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

```
BOOKS : {(title: String,
          authors: [String],
          isbn: Integer,
          year: Date)}

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

Related work

- Kleisli (Buneman, Libkin, Suciu, Tannen, Wong)
- Mnesia/Erlang (Wikström)
- Pdiff (Griffin and Trickey)
- Natural Expert (Hutchison, Neuhaus, Schmidt-Schauss)
- XQuery (Chamberlin, Robie, Wadler, *et al.*)

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>
```

Xduce types

```
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

```
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>
Jérôme Siméon (XML Query WG), Bell Labs, Lucent Technologies <simeon@research.bell-labs.com>

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>

3.3.2 Matches

Notation

The judgment

$$\mathit{Value}\ \textbf{matches}\ \mathit{Type}$$

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.

$$\frac{}{\text{statEnv} \vdash () \textbf{ matches } ()}$$

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

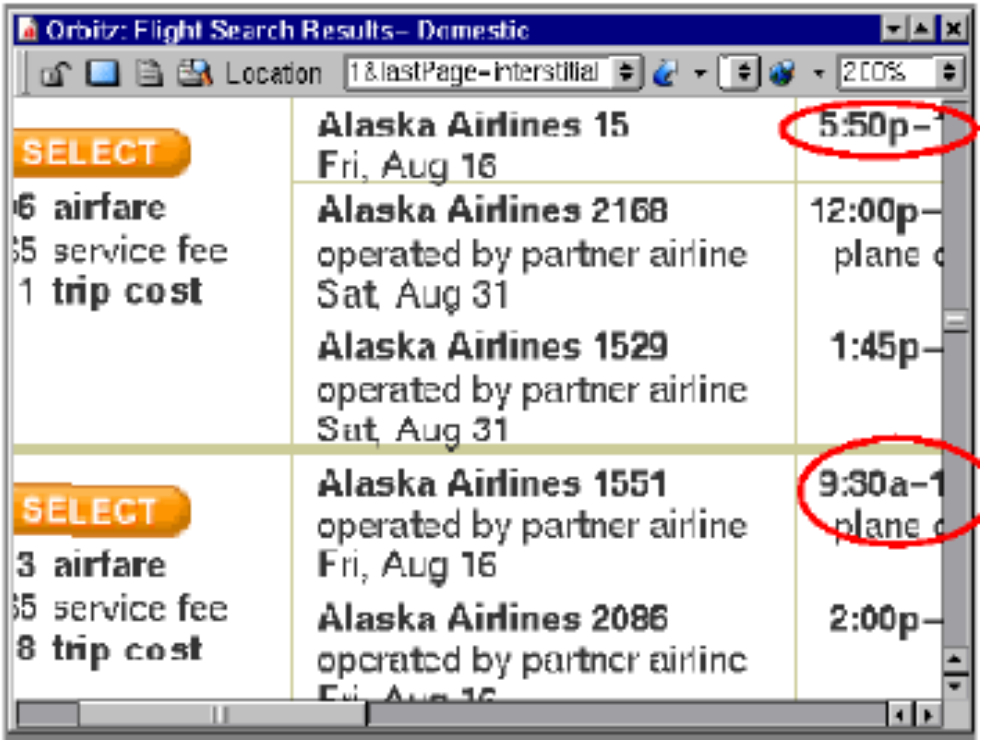
$$\frac{\text{statEnv} \vdash \mathit{Value}_1 \textbf{ matches } \mathit{Type}_1 \qquad \text{statEnv} \vdash \mathit{Value}_2 \textbf{ matches } \mathit{Type}_2}{\text{statEnv} \vdash \mathit{Value}_1, \mathit{Value}_2 \textbf{ matches } \mathit{Type}_1, \mathit{Type}_2}$$

Related work

- Xduce, Xtatic (Pierce, Hasoya, Gapayev, *et al.*)
- Cduce (Castagna, Frisch, *et al.*)
- Bigwig, Jwig (Schwartbach, Møller, *et al.*)
- XQuery (Chamberlin, Robie, Wadler, *et al.*)

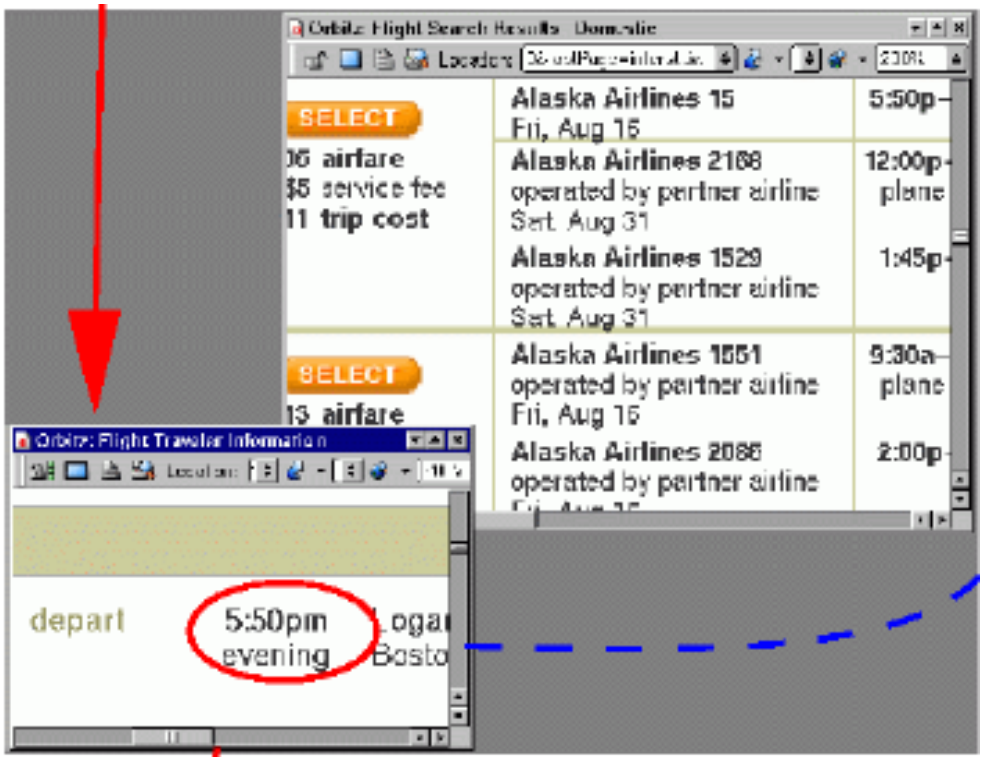
PLT Scheme
Continuations for the Web

Orbitz: Two flights

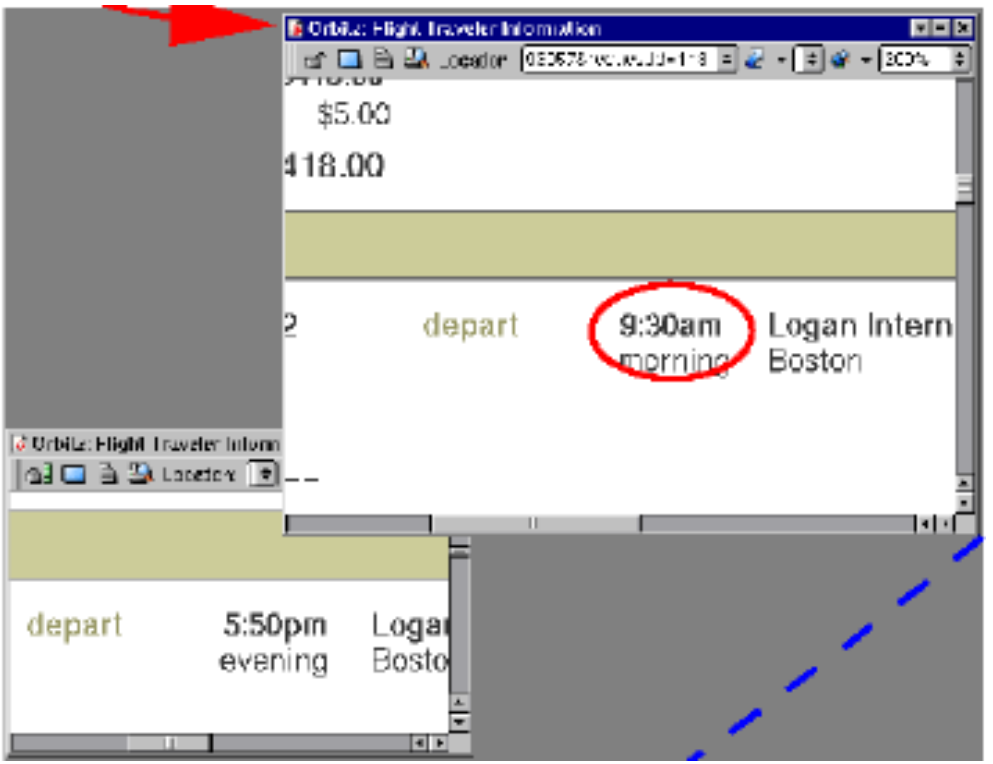


Graunke, Findler, Krishnamurthi, Felleisen (ESOP 2003)

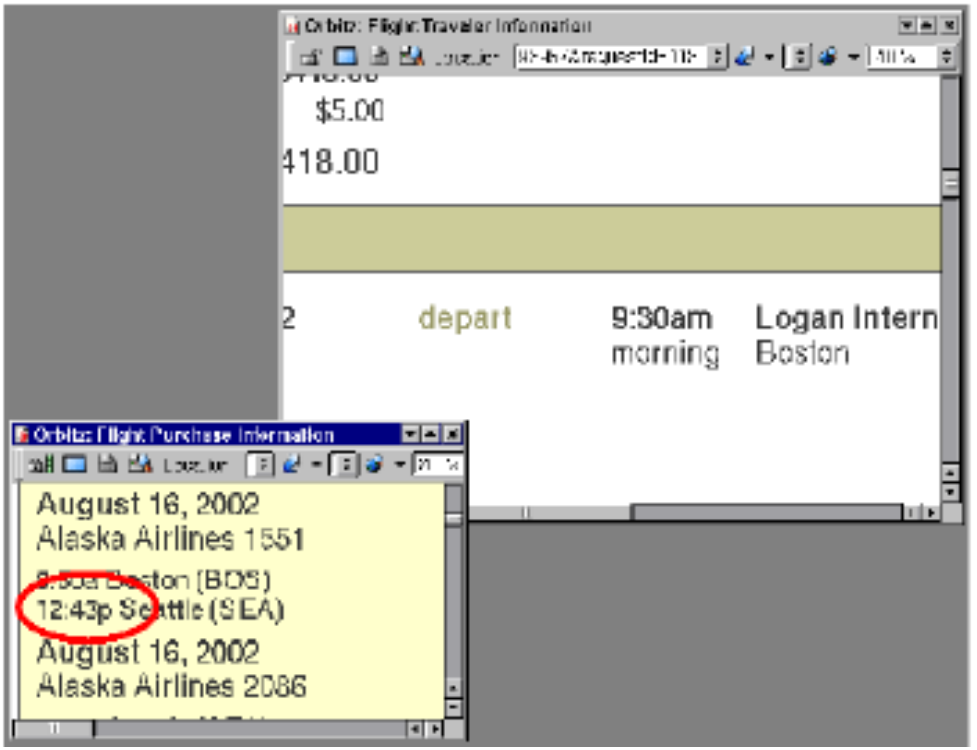
Orbitz: Clone and submit first



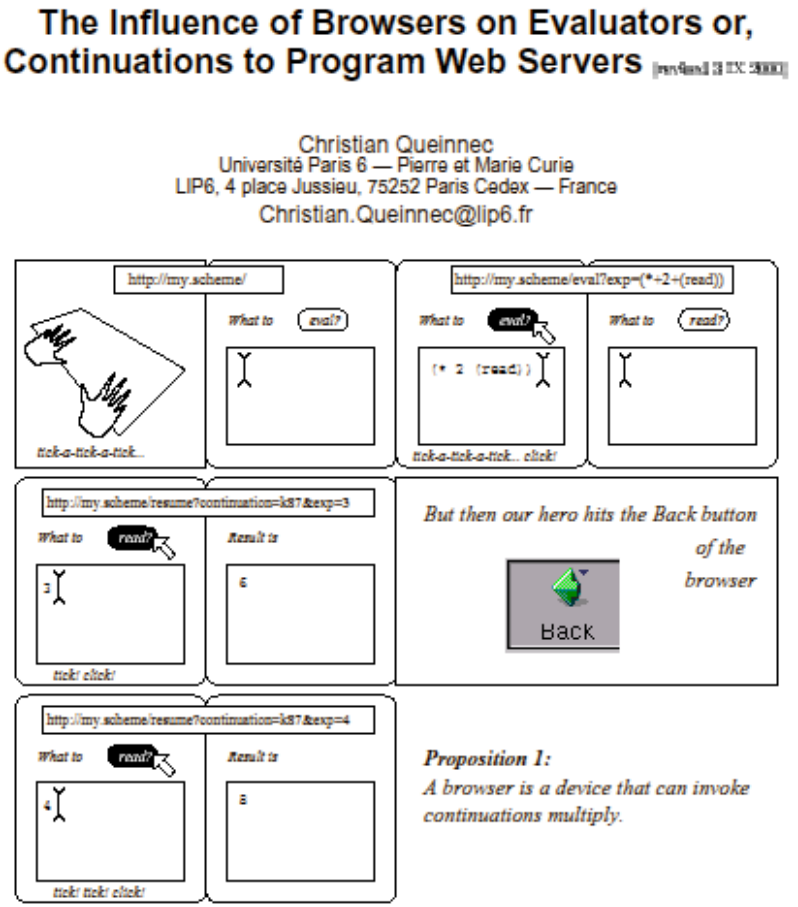
Orbitz: Submit second



Orbitz: Select first – problem!



Quenniec: Browsers and continuations



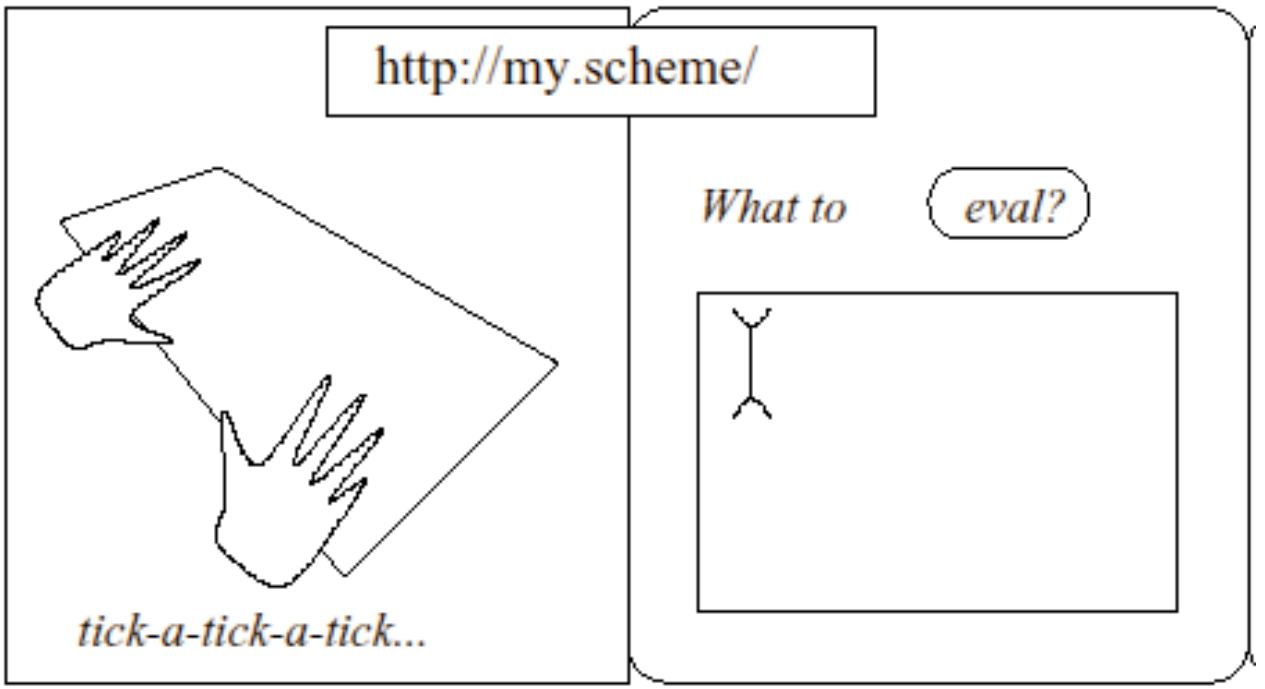
The Influence of Browsers on Evaluators or, Continuations to Program Web Servers

Christian Queinnec
Université Paris 6 — Pierre et Marie Curie
LIP6, 4 place Jussieu, 75252 Paris Cedex — France
Christian.Queinnec@lip6.fr

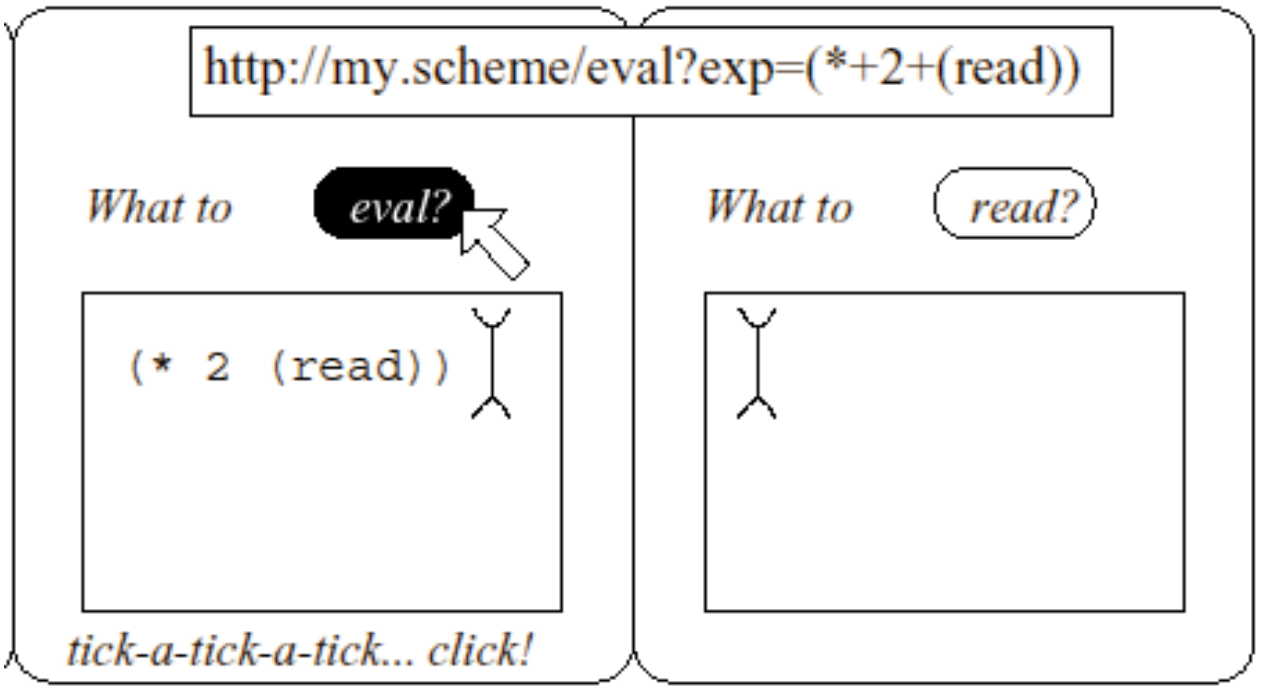
Christian Quenniec (ICFP 2000)

also John Hughes, Paul Graham

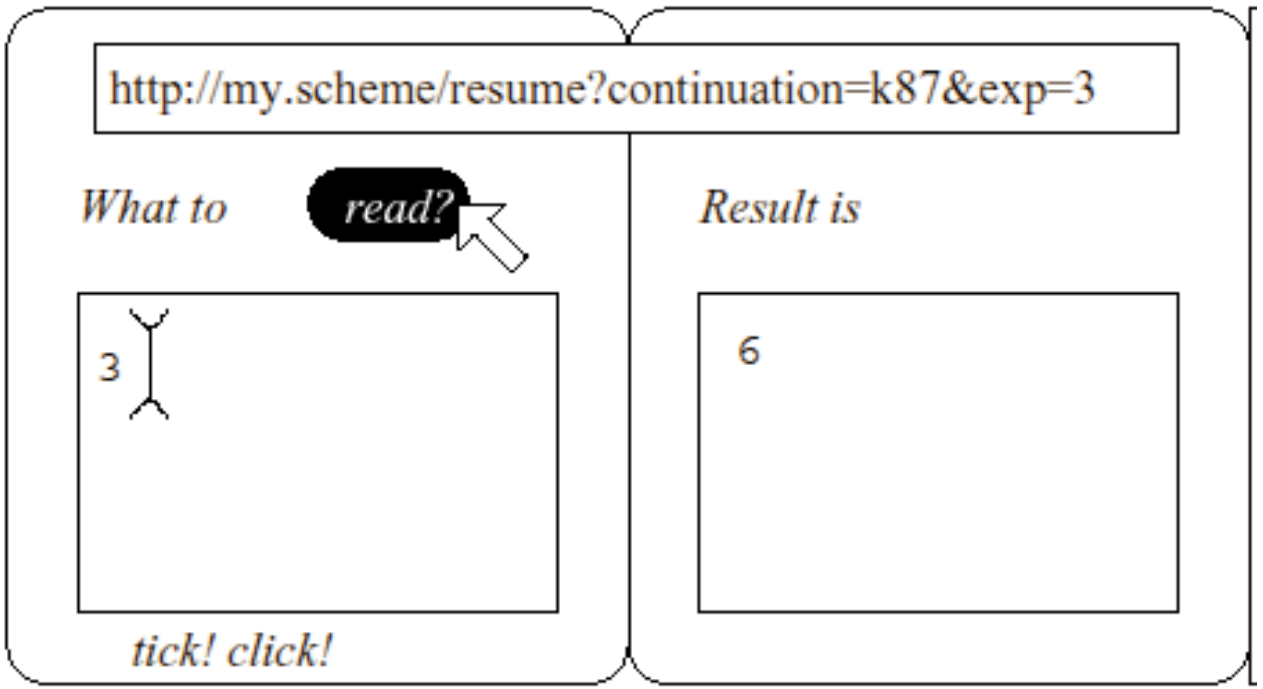
Quenniec: Go to web page



Quenniec: First argument



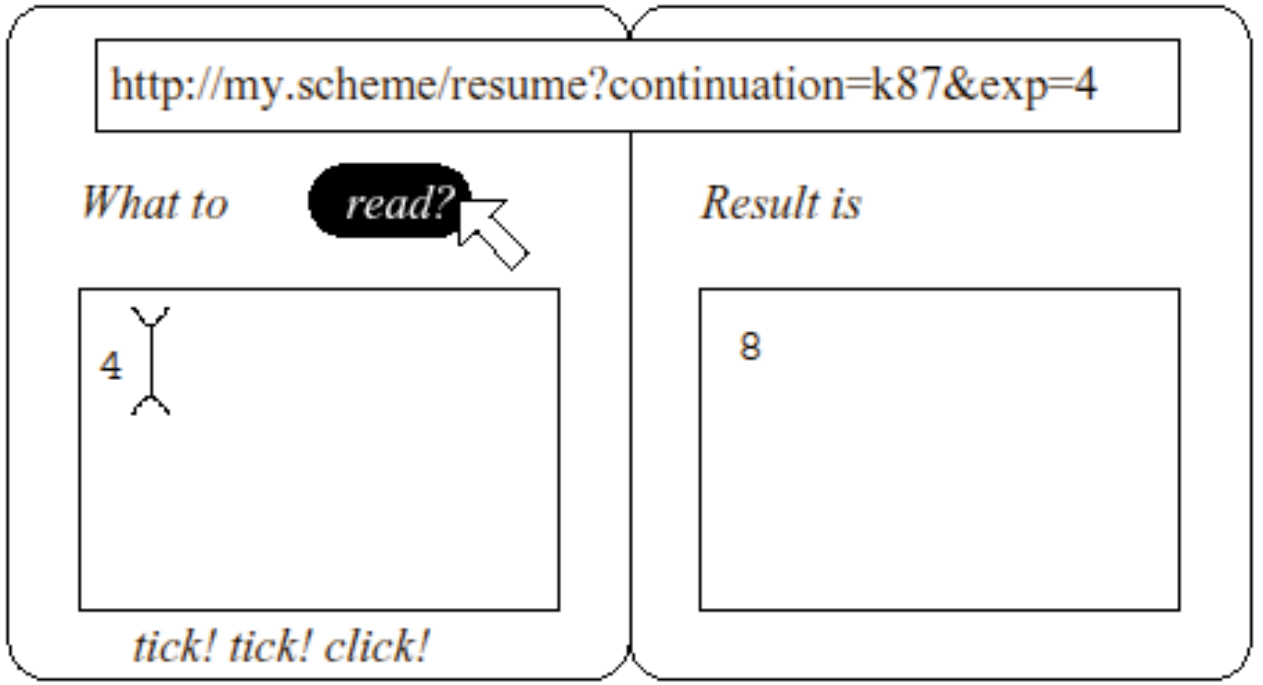
Quenniec: Second argument



Quenniec: Back button

But then our hero hits the Back button of the browser

Quenniec: Second argument, second time



Related work

- Mawl (Ramming, Atkins, Ball, Bruns, Cox)
- Continuations (Quiennec)
- PLT Scheme (Graunke, Findler, Krishnamurthi, Felleisen)
- Bigwig, Jwig (Schwartbach, Møller, *et al.*)
- WASH (Thiemann)

Erlang
Communication via values

Erlang: An area server

```
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.
```

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

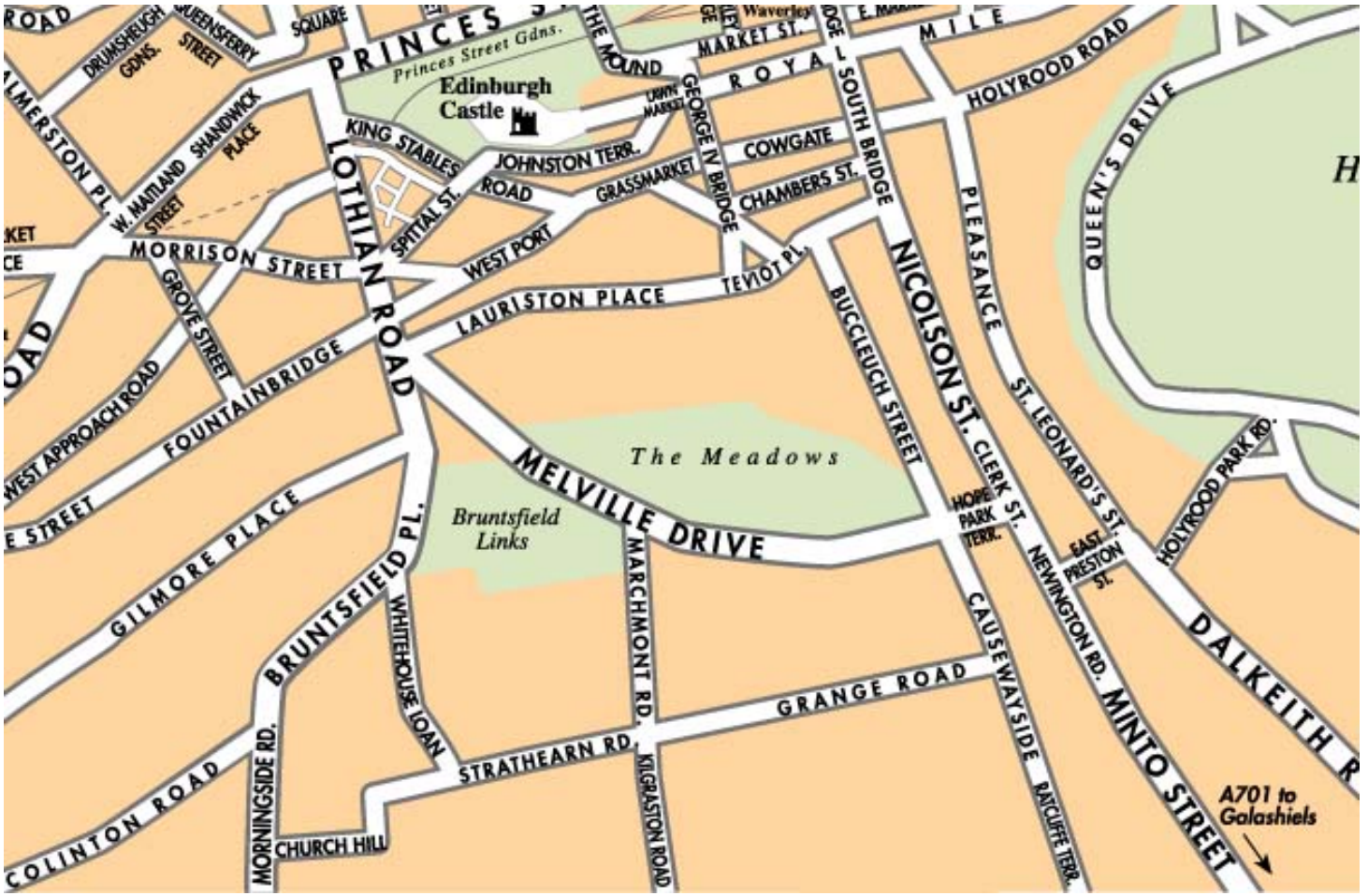
```
start() ->
    start_replicated(area_server, 0, handler/2).
handler({square, X}, Tot) ->
            {X*X, Tot + X*X};
handler({rectangle, [X,Y]}, Tot) ->
            {X*Y, Tot + X*Y};
handler(areas, Tot) ->
            {Tot, Tot}.
```

Related work

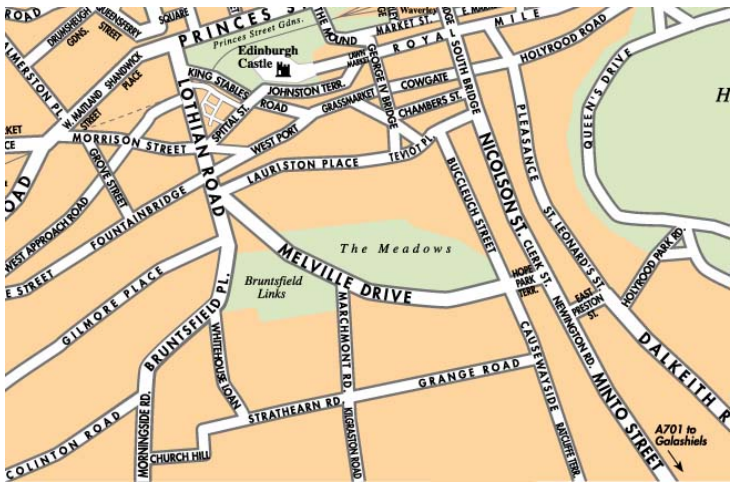
- Erlang (Armstrong, Virding, Wikström, Williams)
- Ensemble (Hayden and vanRenesse)
- Fox (Harper and Lee)
- Plan X (Henglein)

Links

Hope and Links



Hope and Links



Links

(Bruntsfield Links)

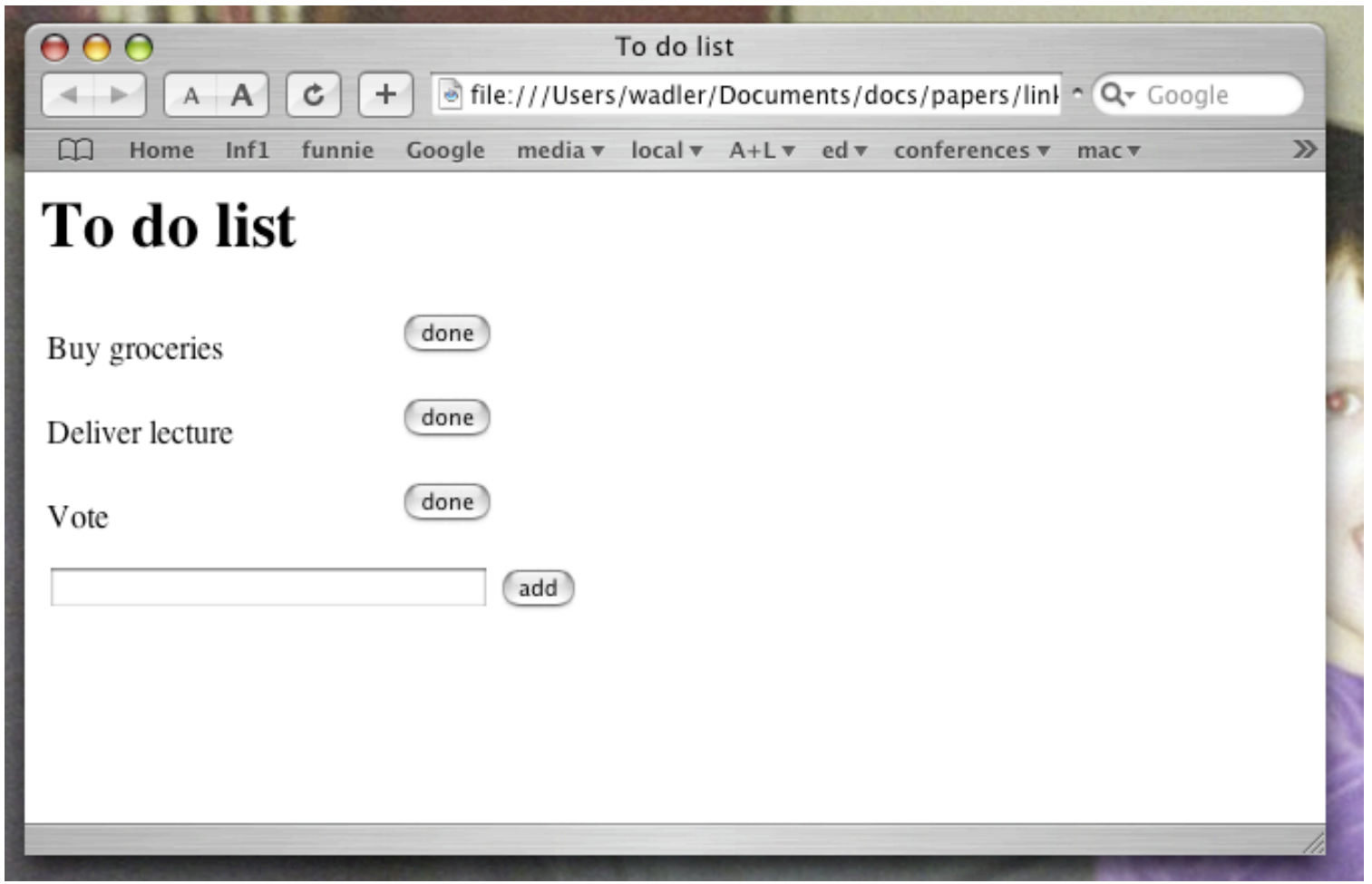
Wadler et al (2005)

Hope

(Hope Park Square)

Burstall, MacQueen, Sannella (1980)

A Links program state in client


To do list
file:///Users/wadler/Documents/docs/papers/link
Google
Home
Inf1
funnie
Google
media
local
A+L
ed
conferences
mac
To do list
Buy groceries
done
Deliver lecture
done
Vote
done
add

```
main() { todo([]) }
todo(items) {
  <html><body>
    <h1>Items to do</h1>
    <table>{
      for item in items return
        <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>
}
```

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>
}
```

An event lookup system

```
table DATES of (date:Date, id:Int)
table EVENTS of (id:Int, name:String, details:String)
main() {
  <html><body>
    <h1>Events</h1>
    <h:form l:action="{events(date)}"
            l:check="{date > today()}">
      <input l:name="{date}" type="text"/>
    </h:form>
  </body></html>
}
```

```
events(d) {
  <html><body>
    <h1>Events on {d}</h1>
    <ul>{
      for (date:d2,id:i) in DATES
        for (id:i2,name:n) in EVENTS
          where d==d2 && i==i2 return
            <li><a l:action="{details(i)}">{n}</a></li>
    }</ul>
  </body></html>
}
details(i) {
  for (id:i2,name:n,details:d) in EVENTS
    where i==i2 return
      <html><body>
        <h1>Details of {n}</h1>
        <p>{ d }</p>
      </html></body>
}
```

Variations

```
events(date) {
  <html><body>
    <h1>Events on {date}</h1>
    <ul>{
      for (=date,id) in DATES
        for (=id,name) in EVENTS
          <li><a l:action="{details(id)}">{name}</a></li>
    }</ul>
  </body></html>
}
details(id) {
  for (=id,name,details) in EVENTS return
  <html><body>
    <h1>Details of {name}</h1>
    <p>{ details }</p>
  </html></body>
}
```

Defining new tags – page template

```
template <page><title>{t}</title>{ c }</page> {
  <html>
    <head><title>{t}</title></head>
    <body>
      <h1>{t}</h1>
      {c}
    </body>
  </html>
}
```

Now I can write

```
details(id) {
  for the (id,name,details) in EVENTS return
    <page>
      <title>Details of {name}</title>
      <p>{ details }</p>
    </page>
}
```

Defining new tags – enter a date

```
template <range from="{from}" to="{to}" {rest}/> {
  <select l:name="{n}" {rest}>{
    for i in [from..to] return
      <option value="{i}">{i}</option>
  }</select>
}
template <date from="{from} to="{to}"/> {
  <range l:name="{year}" from="{from}" to="{to}"
    value="{today().year}"/>
  <range l:name="{month}" from="1" to="12"
    value="{today().month}"/>
  <range l:name="{day}"
    from="1" to="{daysinmonth(month,year)}"
    l:disabled="{!defined(year) || !defined(date)}"/>
}
```

Conclusions

A few open questions

- Syntax? – Can we take a scientific approach?
- Framework for compiling to multiple targets?
- Integrate diverse type systems?
- Monads and effect types?
- Subtyping?
- Type classes and generic programming?
- Integration with Java or .NET?
- Transactions?