

**<bigwig>**

## **Program Analyses for Interactive Web Services**

**Anders Møller**

**Claus Brabrand**

**Anders Sandholm**

**Michael I. Schwartzbach**

 **BRICS**

**Aarhus University**

# Overview

---

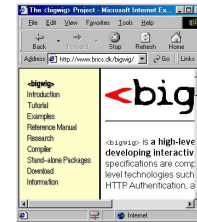
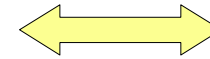
- The **<bigwig>** Project
  - motivation
  - main aspects
  - dynamic documents
  - data-flow analysis
- Gap&Field Analysis [POPL'00]
  - motivation
  - analysis specification
- Valid-HTML Analysis [PASTE'01]
  - summary graph construction and analysis
  - analysis specification
  - future work
- Conclusion

# The <bigwig> Project

Web service construction today:



server



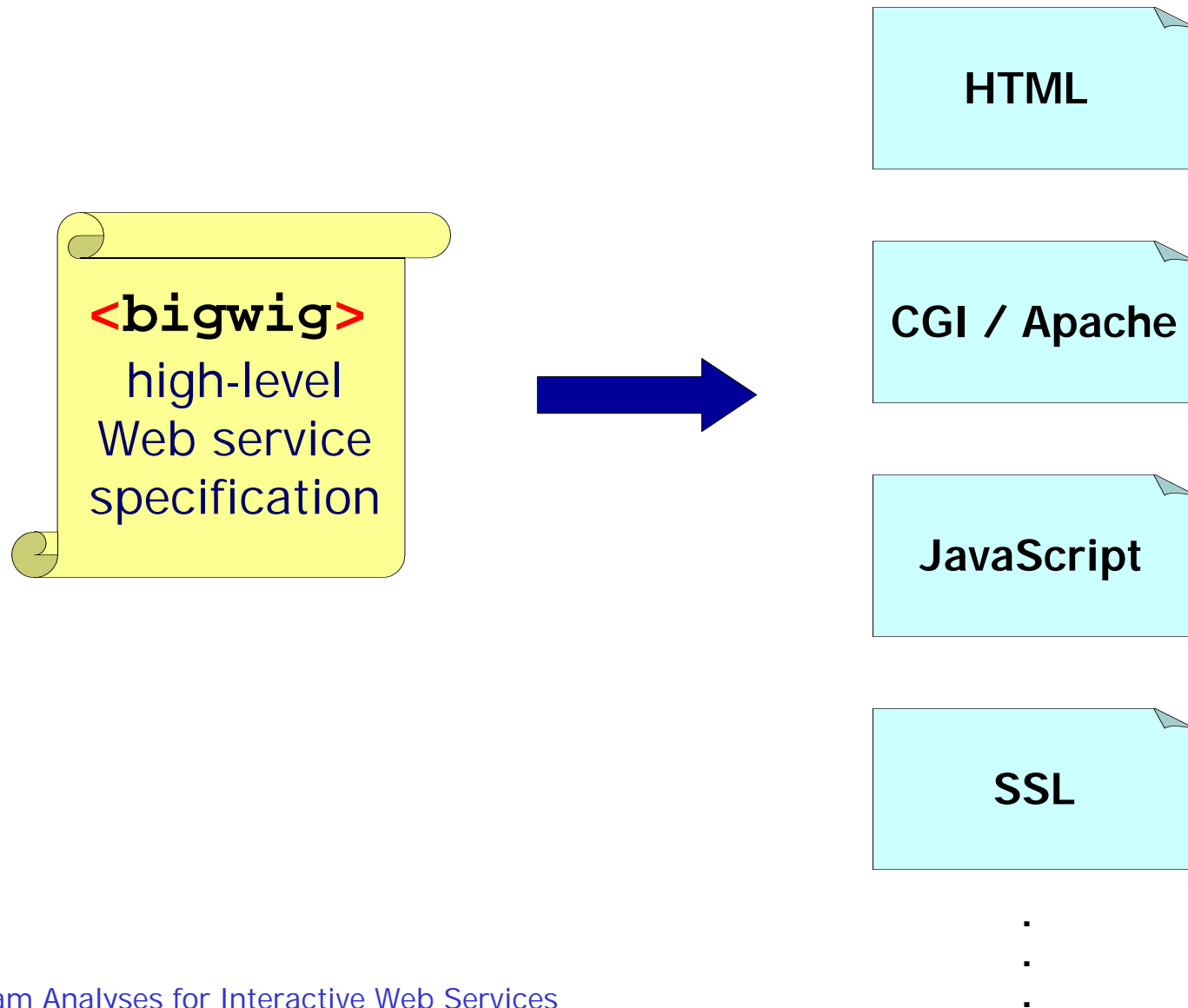
client

- **CGI scripts** (Perl/C)
  - requires knowledge of technical details
  - mixing HTML and code becomes chaotic
- **ASP/PHP/JSP** (embedded scripts, interpreted by server)
  - nice for simple services
  - complex services are hard to maintain
- **Specialized servers** (e.g. Apache modules)
  - full control, efficient
  - low-level, expensive

General problems: *no inherent session concept, no static guarantees!*

# The `<bigwig>` programming language

---



# The `<bigwig>` programming language

---

`<bigwig>` = a type-safe Java/C-like core language

- + an inherent notion of **sessions**
  - + **HTML** fragments as first-class values
  - + a **relational database**
  - + **concurrency control**
  - + declarative **form-field validation**
  - + information-flow and cryptographic **security**
  - + **syntax-level macros**
- } this talk

## Session-based services

---

```
session Hello_1()  
{  
    int d;  
    show <html>Hello World!</html>;  
    d = getWeekDay();  
    if (d==FRIDAY)  
        show <html>You know it's Friday?</html>;  
    else  
        show <html>It's <i>not</i> Friday</html>;  
}
```

- **show** sends an HTML document to client and waits for reply
- *sessions provide **explicit control-flow!*** (not like CGI/PHP/...)

# HTML as a Data Type

---

```
session Hello_2()  
{  
  string s;  
  html H = <html>  
    <h1>Hello World!</h1>  
    Your name: <input type="text" name="f">  
  </html>;  
  show H receive [ s = f ];  
  ...  
}
```

- client input using **input** fields and **show-receive**
- *HTML templates are values!*

# Constructing HTML documents

---

```
session Hello_3()  
{  
  html Cover = <html>  
    <h1>Hello</h1>  
    <[ contents ]>  
  </html>;  
  html Greeting = <html><b>World</b></html>;  
  html H = Cover <[ contents = Greeting ]>;  
  show H;  
}
```

- $x \text{ <[ } g = y \text{ ]}$  plugs  $y$  into  $x$  at the  $g$  gap
- *a highly flexible method for building HTML documents!*

# The challenges

---

Given a **<bigwig>** program, check statically whether

- at every  $x \llbracket g = y \rrbracket$ 
  - $x$  always has a gap named  $g$  ?
  - are  $x$  and  $y$  always “compatible” w.r.t. gaps and fields ?
- at every **show  $x$  receive [  $z = f, \dots$  ]**
  - $x$  always is a *well-formed, valid* HTML document (according to W3C’s specification) ?
  - every input field  $f$  being received always occurs in  $x$  ?
  - the type of  $z$  always matches that of  $f$  ?

Neither Perl, C, Java, PHP, ASP, or JSP can achieve that!

– but **<bigwig>** makes it all possible!

# The solution

---

Apply standard, **data-flow analysis** techniques,  
but with highly **domain-specific lattices**

# A data-flow analysis framework

---

Given

- the **control-flow graph** of the program,
- a finite **lattice** describing the abstract values, and
- monotone **transfer functions** for the various statements,

find the least solution using fixed-point iteration.

The result describes a **conservative approximation** of what values can occur where.

# The *dyndoc* subset of **<bigwig>**

---

## Statements:

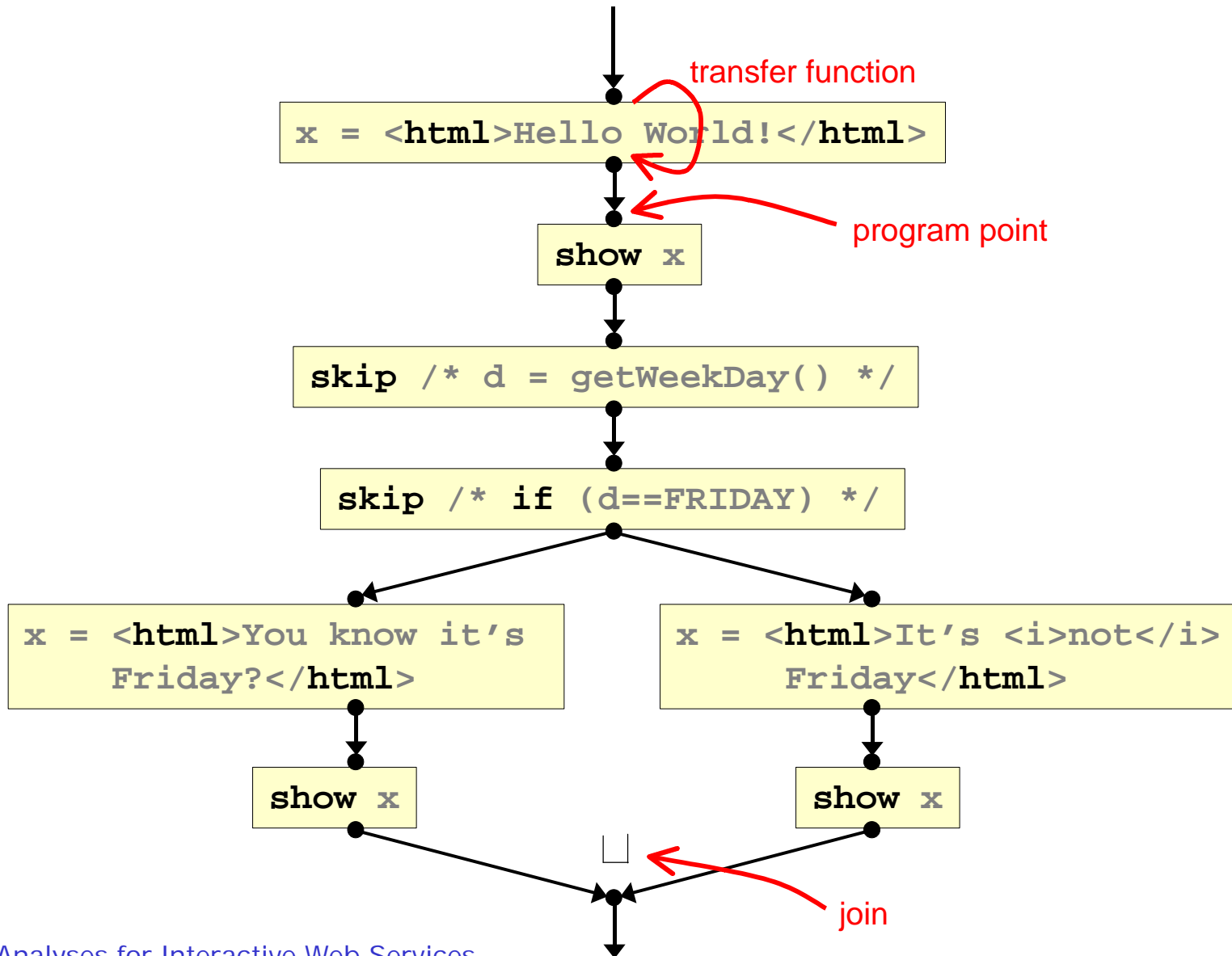
- $x_1 = x_2;$  template variable assignment
- $x = \langle \text{html} \rangle c \langle / \text{html} \rangle$  template constant assignment
- $y = \text{"s"};$  string variable assignment
- $x_1 = x_2 \langle [ g = x_3 ]$  template gap plugging
- $x_1 = x_2 \langle [ h = y ]$  string gap plugging
- **show**  $x$  **receive**  $[ z = f, \dots ]$  client interaction

## HTML fragments:

- $c ::= \langle [ g ] \rangle$  template gap
- |  $\langle \text{tag } a^* \rangle c \langle / \text{tag} \rangle$  element with attribute and contents
- |  $c c$  sequence
- $a ::= n = \text{"s"}$  normal attribute
- |  $n = [ h ]$  attribute gap

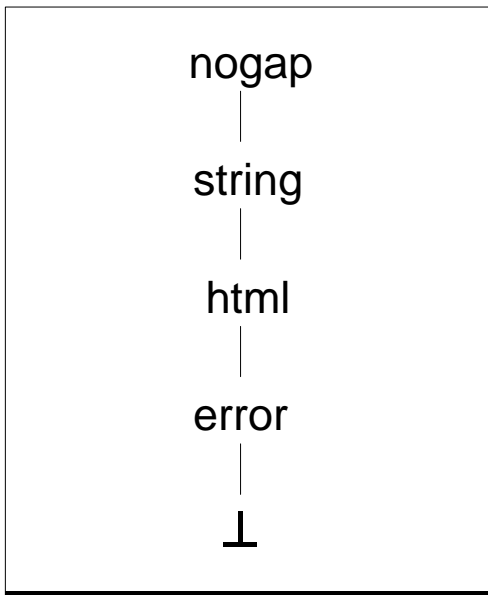
Special HTML elements: **input**, **select**, ...

# A control-flow graph: Hello\_1

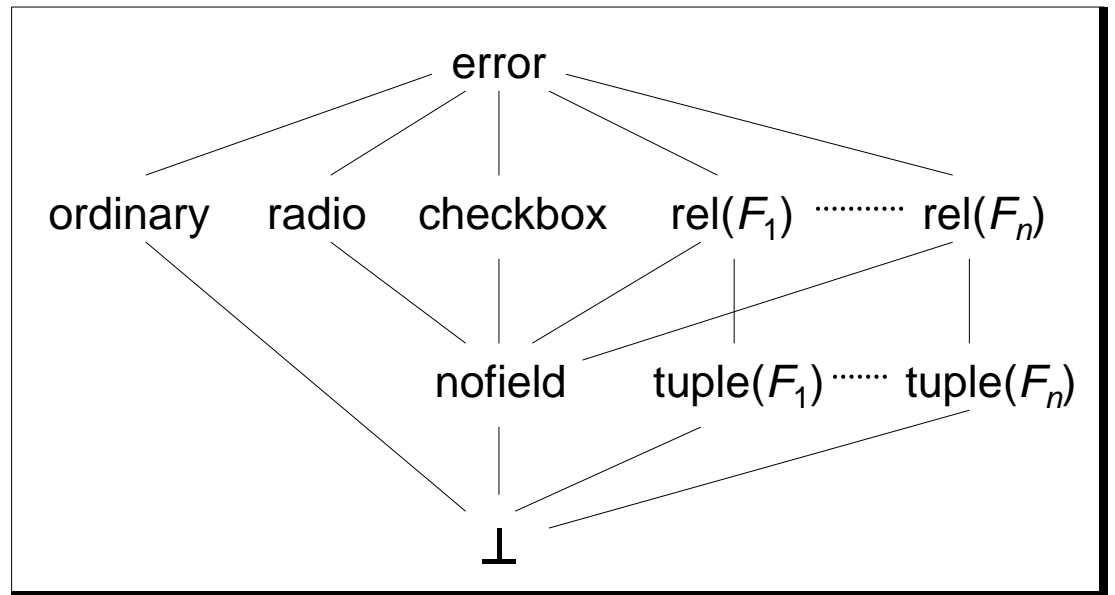


# Lattices for Gap & Field analysis [POPL'00]

GapKind



FieldKind



$Env = HtmlVar \rightarrow (GapName \rightarrow GapKind) \times (FieldName \rightarrow FieldKind)$

Transfer function for statement S:  $T_S: Env \rightarrow Env$

# Valid-HTML analysis [PASTE'01]

---

Two main parts:

1. a data-flow analysis providing a ***summary graph*** for every template variable and program point
2. a validation of each summary graph with respect to an ***abstract description of XHTML***

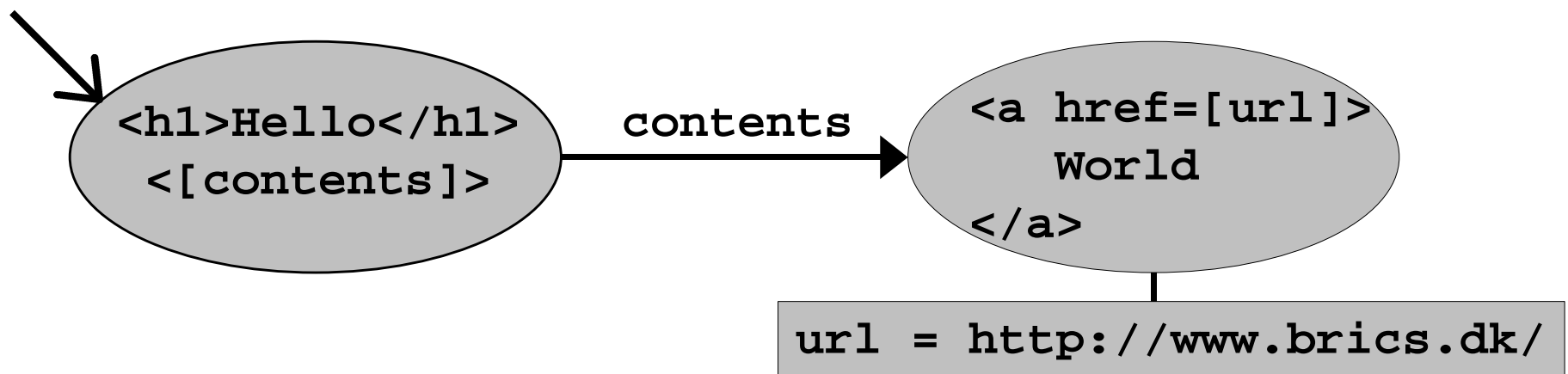
(XHTML is easier to work with than HTML, but is basically the same.)

# Summary Graphs

---

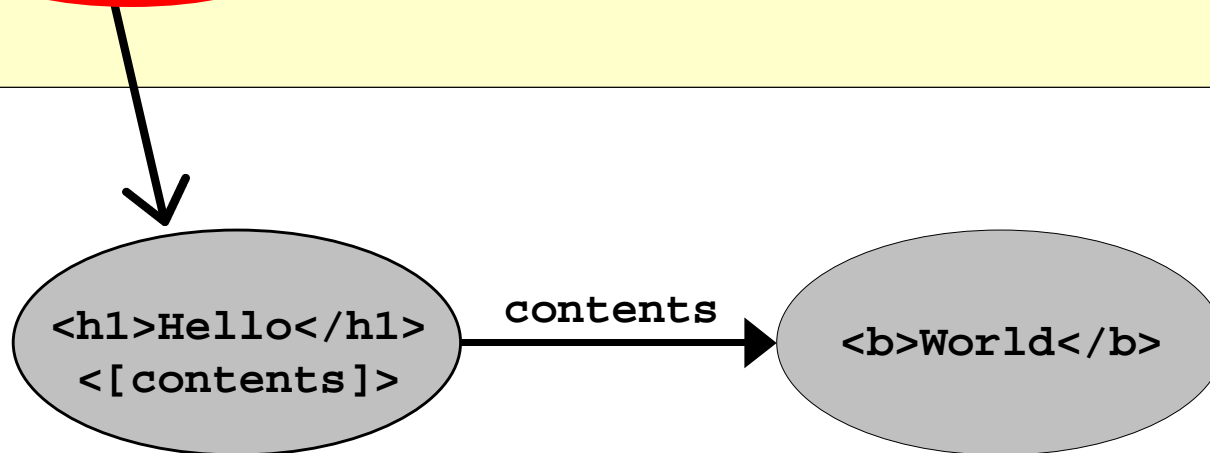
**Summary graph:** a graph representing all the ways templates can be plugged together

- **Node** ~ template constant
- **Edge** ~ template plug
- **Node label** ~ attribute plug
- **Root node** ~ outermost template



# A tiny example

```
session Hello_3()  
{  
  html Cover = <html>  
    <h1>Hello</h1>  
    <[contents]>  
  </html>;  
  html Greeting = <html><b>World</b></html>;  
  html H = Cover <[ contents = Greeting ]>;  
  show H;  
}
```



# Summary Graph inference

---

- The set of summary graphs forms a **lattice**  $(SG, \sqsubseteq)$
- (Quite complex) **transfer functions** can be defined
- A data-flow analysis returns a **result** of the form:

ProgramPoint  $\rightarrow$  HtmlVar  $\rightarrow$  SG

# Transfer function for template plug

---

$$x_1 = x_2 \llbracket g = x_3 \rrbracket$$

is modeled by assigning a new summary graph to  $x_1$  consisting of:

- the union of the **template nodes** from  $x_2$  and  $x_3$
- the **roots** from  $x_2$
- the union of the **attribute labels** of  $x_2$  and  $x_3$
- the union of the **plug edges** in  $x_2$  and  $x_3$  *plus*  
edges from each node in  $x_2$  that may have an open  $g$  gap  
to each root in  $x_3$

# Gap Track analysis

---

– but how do we know which nodes in  $x_2$  that may have open g gaps?

Solution: yet another data-flow analysis

A simple **Gap Track analysis** tracks the origins of gaps:

ProgramPoint  $\rightarrow$  HtmlVar  $\rightarrow$  GapName  $\cup$  FieldName  $\rightarrow 2^{\text{Template}}$

# Validating summary graphs

---

Goal: check that all documents that can be generated from a given summary graph are **valid XHTML 1.0** ( $\approx$  HTML 4.01)

The algorithm:

- traverse the summary graph, compare each node with the requirements in an “abstract DTD”
  - apply memoization
- this phase is both sound and complete!
- bonus: a simple but powerful XML schema language

# Abstract DTDs — a simple schema language

---

An **Abstract DTD** consists of **element** declarations, each containing:

- an element **name**
- **attribute** and **content** declarations
- a **constraint** : a boolean expression of
  - **attr**(a) - "require attribute presence"
  - **content**(c) - "require content presence"
  - **order**( $\{c_1, \dots, c_n\}, \{c_1', \dots, c_m'\}$ ) - "require content ordering"
  - **value**(a, *regexp*) - "require attribute value"

defining a **set of valid XML trees** in a top-down manner.

(Examples: XHTML, WML, VoiceXML, ...)

## Experiments: Valid-HTML analysis

---

Program	Lines	Fragments	Time (sec.)
chat	65	3	0.1
guess	75	6	0.1
calendar	77	5	0.1
xbiff	561	18	0.1
webboard	1132	37	0.6
cdshop	1709	36	0.5
jaoo	1941	73	2.4
bachelor	2535	137	8.2
courses	4465	57	1.3
eatcs	5345	133	6.7

( after macro expansion )

( Linux / 800MHz Pentium III )

– many validation errors found, no spurious errors!

# Example HTML validation report

---

```
--- brics.wig:24: HTML validation:
brics.wig:4:
  warning: illegal attribute 'bgcolo' in 'body'
  template: <body bgcolo=[color]><form>...</form></body>

brics.wig:5:
  warning: possible illegal subelement 'td' of 'table'
  template: <table><[contents]></table>
  contents: td
  plugs: contents:{brics.wig:22}

brics.wig:10:
  warning: possible element constraint violation at 'br'
  template: <br clear=[clear]/>
  constraint: value(clear,{left,all,right,clear,none})
  plugs: clear:{brics.wig:23}
```

# *“Why not just use standard PL techniques?”*

---

*“An XML document is just a **tree**, so simply*

- use **recursive types** instead of **abstract DTDs**, and*
- use **constructor functions** instead of the **gap/plug** approach”*



## *“Why not just use standard PL techniques?”*

---

### **Bad idea!**

- **abstract DTDs** are *strictly more expressive* than **recursive types**
    - example: the <title> element in <head> in HTML
  - the **gap/plug** approach is *strictly more expressive* than **constructor functions**
    - we can do “non-local” plugs instead of building the trees bottom-up only
- and we need this extra power in practice!

# Summary

---

- Increasing need for high-level programming language for making interactive Web services
- Two central ideas in **<bigwig>**:
  - the session model, show/receive
  - flexible but safe, dynamic construction of Web pages:
    - gap/plug document construction
    - gap&field safety
    - always showing valid HTML

# Summary

---

- Gap analysis
- Field analysis
- HTML analysis
  - Gap Track analysis
  - Summary Graph inference
  - Summary Graph validation wrt. XHTML DTD

data-flow  
analyses

summary graph  
analysis

## Papers from the `<bigwig>` project

---

- *A Runtime System for Interactive Web Services* [WWW8]
- *Distributed Safety Controllers for Web Services* [FASE'98]
- *PowerForms: Declarative Client-Side Form Field Validation* [WWW Journal]
- *Growing Languages with Metamorphic Syntax Macros* [submitted]
- *Language-Based Caching of Dynamically Generated HTML* [submitted]
- *The `<bigwig>` Project* [submitted]
- *A Type System for Dynamic Documents* [POPL'00]
- *Static Validation of Dynamically Generated HTML* [PASTE'01]

# Conclusion

---

## Future work:

- full XML support (with schemas as types)
- support for Web site construction
- *JavaWig*: integrating **<bigwig>** into Java
- other aspects of Web programming...

## More information:

**<http://www.brics.dk/bigwig/>**

- Open Source implementation
- documentation and examples
- research papers