

# Type Checking with XML Schema in XACT

Christian Kirkegaard  
Anders Møller



University of Aarhus



# Goals of the XACT Project

---

- Integration of **XML** into general purpose programming languages (**Java**)
- High-level, flexible programming style
- Static guarantees of XML validity
- Efficient runtime model

# The Design of XACT

---

XML data is represented as *templates*

- well-formed XML fragments
- contain **gaps** (named / expressions)
- **immutable**

```
XML x =
  [[ <h:html>
    <h:head>
      <h:title>< [TITLE] ></h:body>
    </h:head>
    <h:body bgcolor={color}>
      <h:h1>< [TITLE] ></h:h1>
      < [NAME] >
    </h:body>
  </h:html> ]];
```

# XML Operations in XACT

---

- **constant** – constructs template from constant string
- **plug** – inserts templates or strings into gaps
- **select** – selects subtemplates (using **XPath**)
- **gapify** – converts subtrees to gaps
- **get** – imports XML tree
- **cast** – runtime check of validity
- **analyze** – *compile-time* check for validity
- **toString** – exports XML tree
- ...

# Validity Analysis

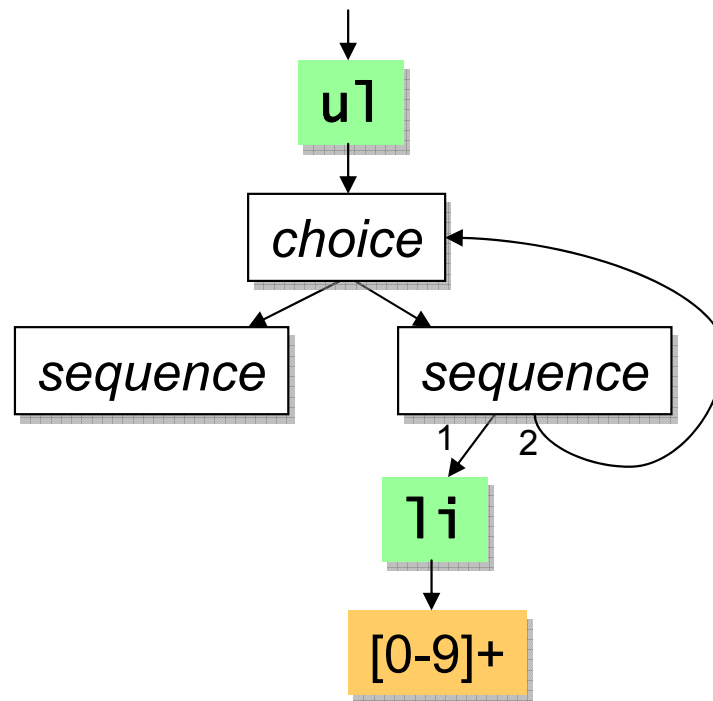
---

- Data-flow analysis
  - using lattice of *summary graphs*
  - transfer functions:
    - abstract **XPath** evaluation on summary graphs
    - conversion from **template constants** to summary graphs
    - conversion from **schemas** to summary graphs
- Validation of summary graphs

# Summary Graphs

---

- A summary graph represents a **set of XML templates** (like an “XML tree” with loops, choices, named gaps, and regexp chardata)
- *Example:* **u1** lists with zero or more **1i** items that each contain an integer



# What's New?

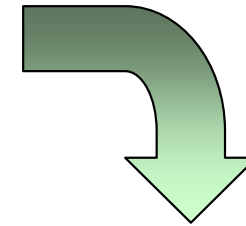
---

- **XML Schema**  
(instead of DTD)
- **Optional type annotations**  
(instead of whole-program analysis)

# Example

---

```
<cardlist xmlns="http://businesscard.org">
  <card>
    <name>John Doe</name>
    <email>john.doe@widget.inc</email>
    <phone>(202) 555-1414</phone>
  </card>
  <card>
    <name>Zacharias Doe</name>
    <email>zach@notmail.com</email>
  </card>
  <card>
    <name>Jack Doe</name>
    <email>jack@mailorder.edu</email>
    <email>jack@geemail.com</email>
    <phone>(202) 456-1414</phone>
  </card>
</cardlist>
```



## My Phone List

- John Doe, phone: (202) 555-1414
- Jack Doe, phone: (202) 456-1414

# Example

---

```
import dk.brics.xact.*;
import java.io.*;

public class PhoneList {
    @Namespace static final String b = "http://businesscard.org";
    @Namespace static final String h = "http://www.w3.org/1999/xhtml";
    @Namespace static final String s = "http://www.w3.org/2001/XMLSchema";

    XML<h:html[s:string TITLE, h:Flow MAIN]> wrapper;

    private void setDefaultwrapper(String color) {
        wrapper =
            [[ <h:html>
                <h:head>
                    <h:title><[s:string TITLE]></h:title>
                </h:head>
                <h:body bgcolor={color}>
                    <h:h1><[s:string TITLE]></h:h1>
                    <[h:Flow MAIN]>
                </h:body>
            </h:html> ]];
    }
}
```

```

public static void main(String[] args) {
    XML<h:html> x = new PhoneList().transform(args[0]);
    System.out.println(x);
}

XML<h:html> transform(String url) {
    XML cardlist = XML.get(url, "b:cardlist");
    setDefaultWrapper("white");
    return wrapper.plugin("white", cardlist);
}

XML<h:url> makeList(XML cardlist) {
    XML r = [[ <h:url><[CARD]>
    for (XML c : x.select("b:cardlist"))
        r = r.plugin("CARDS",
            [[ <h:li>
                <h:b><{c.select("b:name/text()")}></h:b>,
                phone: <{c.select("b:phone/text()")}>
            </h:li>
            <[CARDS]> ]]);
    return r;
}
}

```

**XACT analysis:**

**VALID!**

# XML Schema and Restricted RELAX NG

---

- **Restricted RELAX NG:** a subset of RELAX NG, suitable as intermediate language for working with XML Schema
  - sufficient for exact and simple embedding of XML Schema
  - tractable summary graph validation
- (RELAX NG: schema language closely related to regular tree grammars)

# Restricted RELAX NG

---

Restrictions:

**[single-type grammar]** - as XML Schema

**[attribute context insensitivity]** - no choices of attributes

**[interleaved content]** - interleave limited to top-level content model descriptions

# XML Schema → Restricted RELAX NG

---

- Mostly straightforward...
- all → interleave
- wildcards → name classes
- Type derivation by extension
- Nillable elements
- Substitution groups

- Assume element  $E$  has type  $T$ , and  $T'$  is derived by extension from  $T$
- `xsi:type="T"` or `xsi:type="T' "` in instance documents
- ***Internally, encode `xsi:type` in the element name!***
- All references to  $E$  use a choice of  $E@T$  and  $E@T'$

- `xsi:nil="true"`
- ***Internally, encode `xsi:nil` in the element name!***
- All references to  $E$  use a choice of  $E$  and  $E\%nil$

- Assume element  $F$  is in substitution group of  $E$
- All references to  $E$  use a choice of  $E$  and  $F$

# Restricted RELAX NG to Summary Graphs

---

- A summary graph is almost just a **graphical representation** of a Restricted RELAX NG schema
- Slightly extended definition of summary graphs:
  - element/attribute names are now regular string languages
  - introducing *interleave* nodes (variation of *sequence* nodes)
- XML Schema Datatypes → regular string languages

# Validating Summary Graphs

---

Given a summary graph node  $N$  and  
a Restricted RELAX NG pattern  $P$ ,  
is  $L(N) \subseteq L(P)$  ?

# Validating Summary Graphs

---

1. Construct a (small) **context-free grammar**  $G_N$  for  $N$ , treating element and text nodes as terminals (and ignoring attributes and contents of elements)
2. If  $G_N$  is non-linear, make upper approximation to a **regular** grammar
3. Construct a (small) **regular grammar**  $G_P$  for  $P$ , treating element, text, data, and value patterns as terminals (and ignoring attributes and contents of elements)
4. Encode each element terminal as a string:  $\langle name \rangle$
5. Encode each text node and each text/data/value pattern as a regular grammar (or a finite-state automaton)
6. Check  $L(G_N) \subseteq L(G_P)$  (*now regular string language inclusion!*)
7. For each element node  $E_N$  in  $G_N$ , find an element pattern  $E_P$  in  $G_P$  with matching name class for each possible name of  $E_N$
8. Check that the attributes of  $E_N$  match those of  $E_P$  ...
9. Check, **recursively**, that the content model of  $E_N$  matches that of  $E_P$

# What about *interleave*?

---

- Limited to top-level content model descriptions
- Sub-patterns must be disjoint
- If  $P$  is an *interleave* pattern,
  - test each sub-pattern in turn, projecting onto its element names
  - check that all element names  $G_N$  also occur in one of the sub-patterns
- If  $N$  is an *interleave* node
  - generalized product construction...

# Optional Type Annotations

---

- Old XACT:
  - whole-program data-flow analysis
  - schema type annotations at input/output only
- New XACT:
  - optional schema type annotations at XML types
  - more modular analysis
  - typed gaps in templates:  
`XML<h:html[s:string TITLE, h:Flow MAIN]>`
  - new analysis of reads, new check for writes

# Related Work

---

- **XJ**
  - mandatory vs. optional type annotations?
  - mutable vs. immutable XML data?
  - nominal vs. semantic subtyping?
- **XDuce/Xtatic/CDuce**
  - regular expression types vs. summary graphs
  - regular expression types vs. XML Schema?
- **C $\omega$** 
  - hiding vs. embracing XML?
- **XSLT analysis**

# Conclusion

---

- XACT now supports **XML Schema** and **optional type annotations**
- **Restricted RELAX NG** is a useful intermediate language for validating with XML Schema
- **Optional type annotations** make it possible to balance between *flexibility* and *modularity*