Static Analysis for Java Servlets and JSP

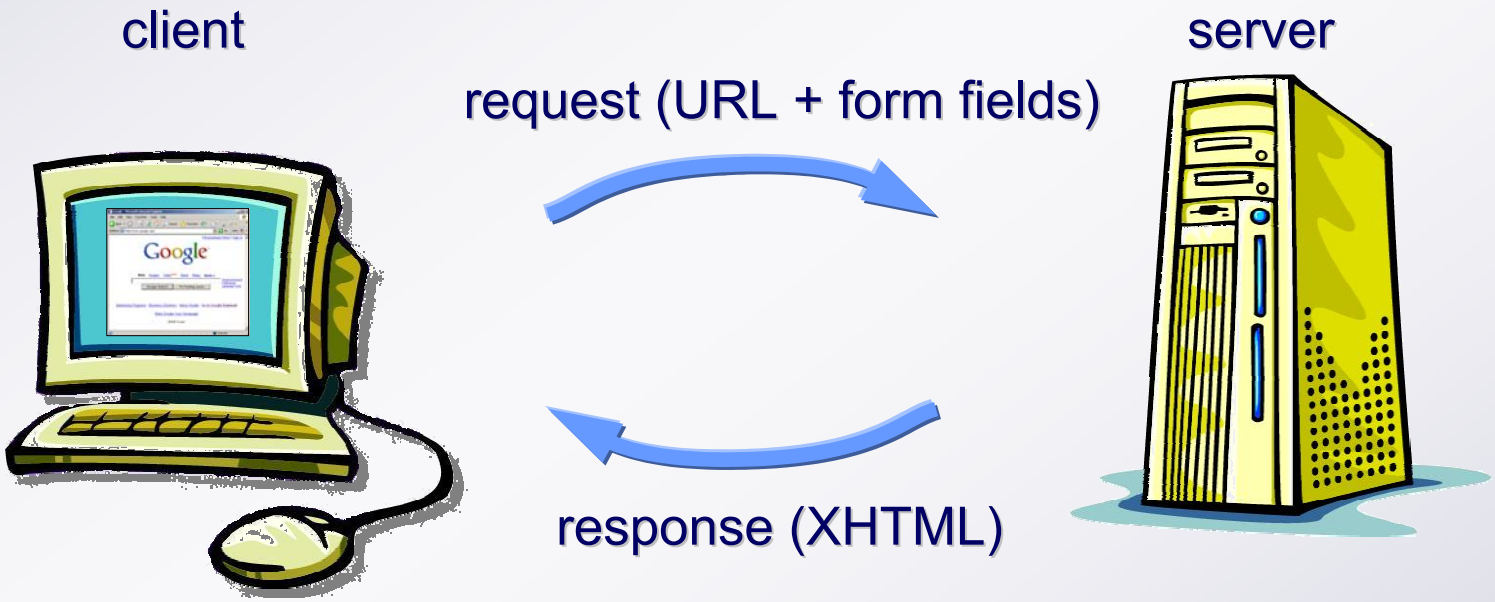
Christian Kirkegaard

Anders Møller

BRICS, University of Aarhus

Java Servlets and JSP

- A powerful framework for **Web application** development
- Communication using **HTTP**:

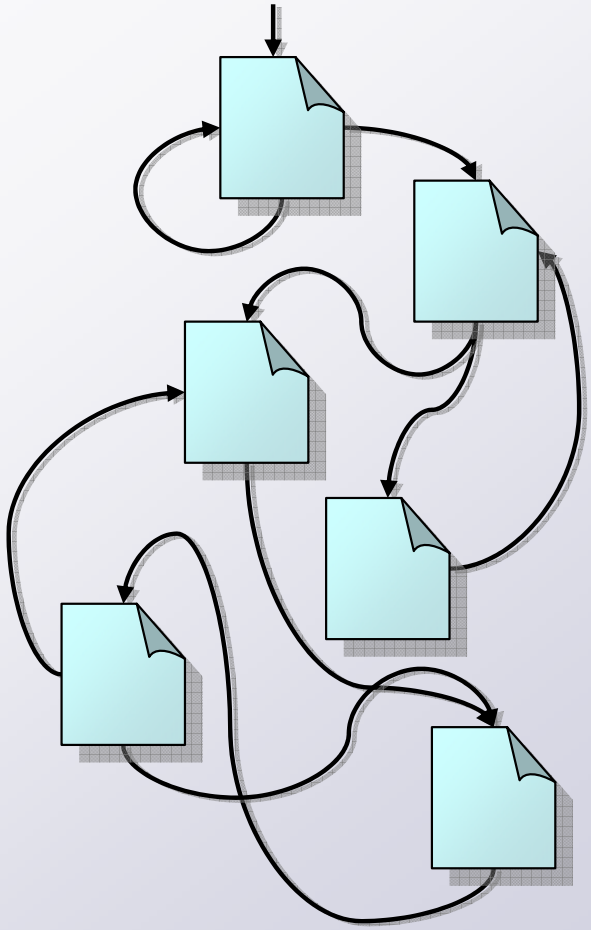


- JSP pages are compiled into servlets...

The Servlet API

- A **Web application** contains a collection of **servlets**
- A configuration file maps URL requests to servlet names
- Each **servlet** receives **user input** (form fields) and produces **XHTML** output

- *The output is generated by* ***printing characters to a stream!***

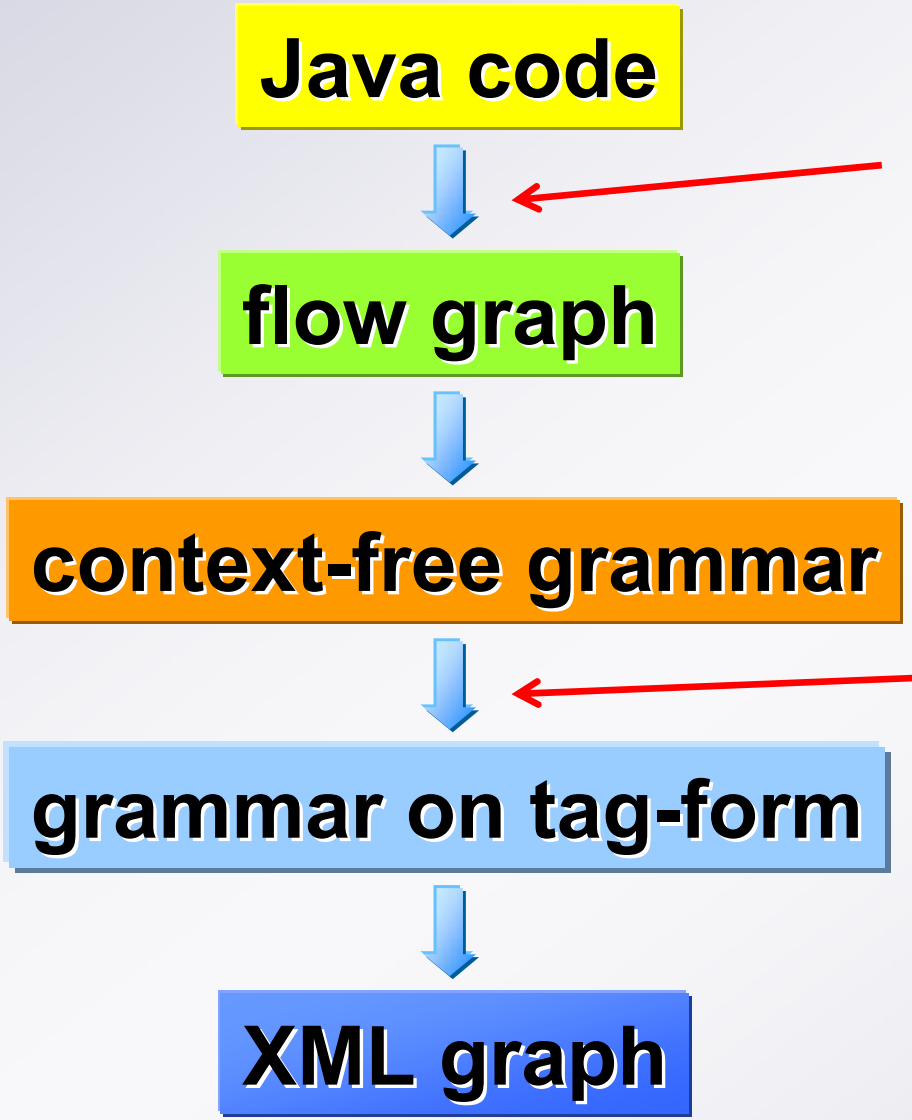


Two challenges for Web app developers

1. *Is the response always **well-formed** and **valid**?*

2. *Do **forms and fields** being generated always match the code that receives the input?*

Overview of the analysis



Soot (Hendren et al.)
+ string analysis (SAS'03)

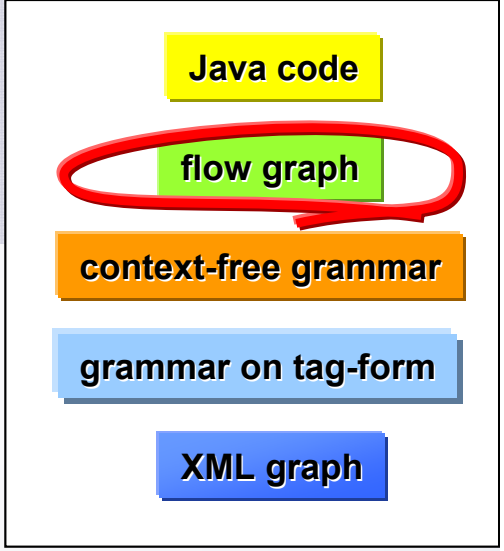
balanced grammars (Knuth)
+ much more...

XML Schema validation (PLAN-X'06)

Our results

- Construction of **context-free grammars** that approximate the possible output on **output streams in Java**
- Checks for XML **well-formedness** and XML Schema **validity** on **context-free grammars**
- **Inter-servlet control flow** for Web apps

Flow graphs

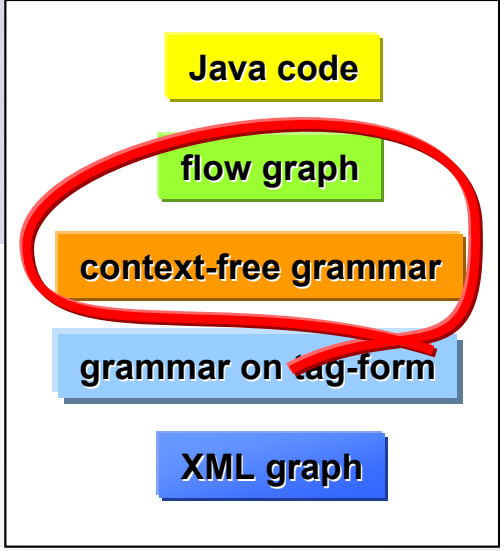


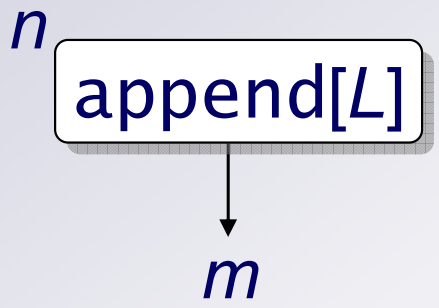
- *Nodes:*
  - `append[`*regular language*`]`
  - `invoke[`*possible targets*`]`
  - `nop`
  - `return`
- *Edges* represent control flow

– **Soot** (Hendren et al.) gives us Jimple code, control flow analysis, and alias analysis

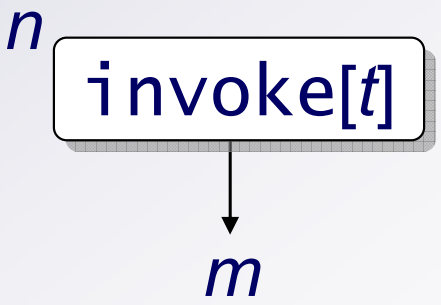
– **String analysis** (SAS'03) gives us a regular language for each string expression

Flow graphs → context-free grammars

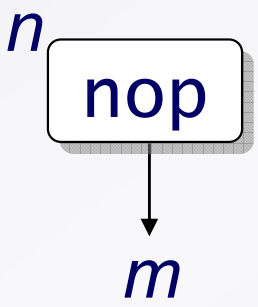




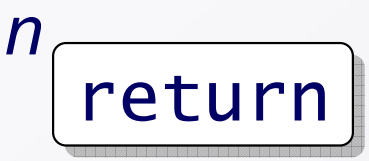
$$N_n \to R_L\, N_m$$



$$N_n \to N_t\, N_m$$

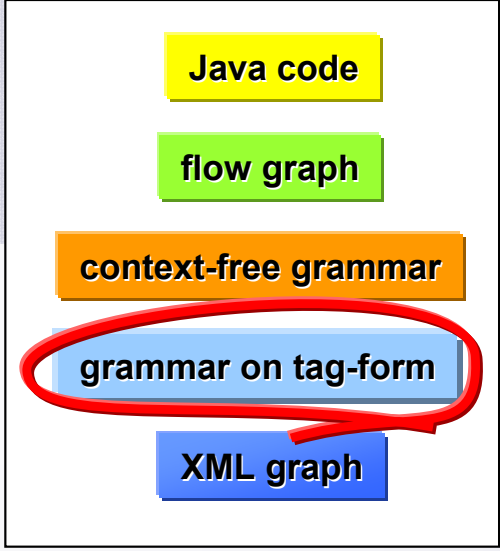


$$N_n \to N_m$$



$$N_n \to \varepsilon$$

Tag-form

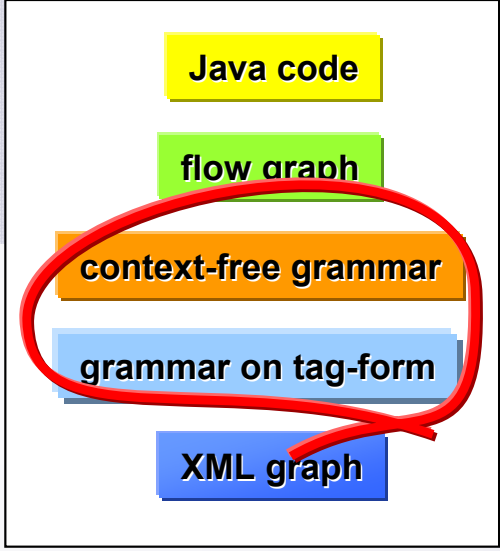


- $C \rightarrow \langle\ T\ A\ \rangle\ C\ \langle/\ T\ \rangle$ (element)
- $C \rightarrow X$ (text)
- $C \rightarrow C\ C$ (content sequence)
- $A \rightarrow W\ T = ”\ V\ ”$ (attribute)
- $A \rightarrow A\ A$ (attribute sequence)
- $A \rightarrow \varepsilon$ (empty attr. seq.)

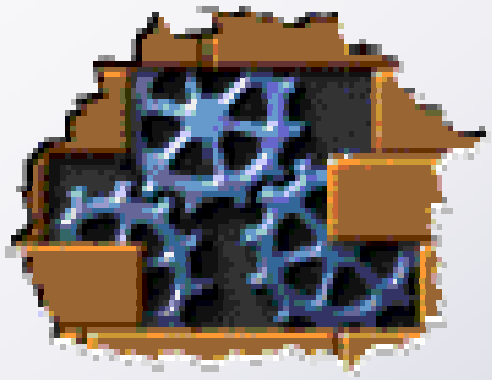
– if the grammar can’t be brought on this form, its language is not **well-formed XML**

– on tag-form, we can easily check the remaining properties for well-formedness...

Transforming to tag-form

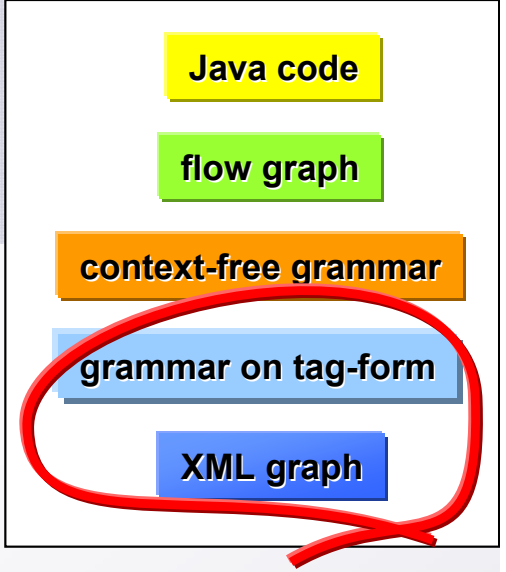


1. make </ a single symbol
2. obtain **balanced grammar** with respect to < and </
   - *Knuth '67*
   - *Mohri & Nederhof '01*



3. obtain unique contexts (tag / element content / attribute value)
   – if possible
4. apply standard grammar transformations (inlining, expansion)...

Checking validity with XML graphs



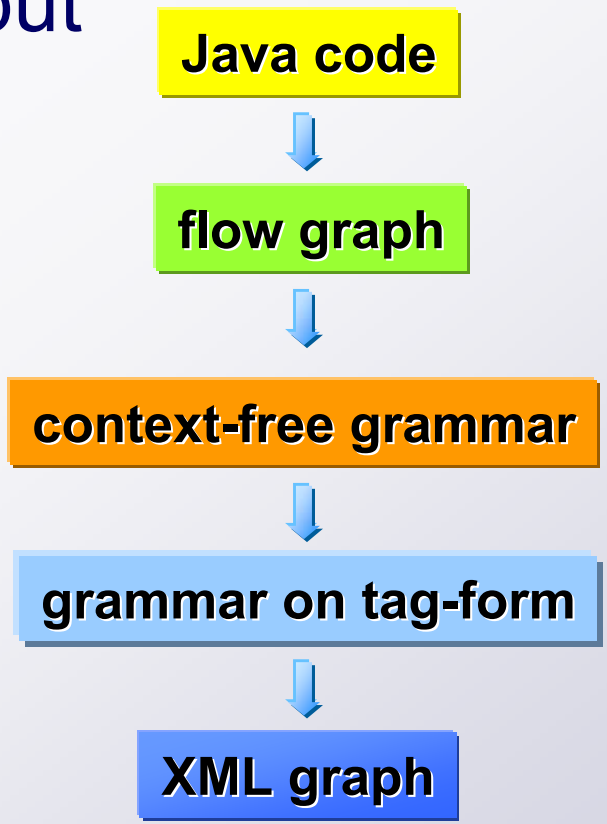
- An **XML graph** is like an XML tree but with
  - *choices* and *loops*
  - attribute/element names and text described with *regular string languages*

  (so one XML graph represents a *set* of concrete XML trees)

- Converting a context-free grammar on tag form into an XML graph is straightforward
- We have a tool for **validating** XML graphs relative to schemas (using **XML Schema**)!

Conclusion

- Construction of **context-free grammars** that approximate the possible output on **output streams in Java**
- *Sound and complete* checks for XML **well-formedness** and XML Schema **validity** on **context-free grammars**
- **Inter-servlet control flow** for Servlets/JSP Web apps



- ...and the implementation is on the way – reeeal soon now ☺