Combinatorial Test Set Generation: Concepts, Implementation, Case Study

Drs. Vadim V. Zaytsev

22 June 2004
Legal stuff

- **Supervisor**: Prof.dr. Hendrik Brinksma, UT

- **Ext.Supervisor**: Dr.ing. Ralf Lämmel, VU&CWI

- **Hosting organisation**: Vrije Universiteit Amsterdam

- Contributes to a collaboration between Dr. Wolfram Schulte from MSR/FSE and Dr.ing. Ralf Lämmel from VU&CWI (**Geno** project).
Contents of the project

- Testing
  - Combinatorial test data generation
  - Differential approach

- Grammarware
  - XML Schema as grammar description formalism
  - XML validators as grammar-based software

http://www.cs.vu.nl/grammarware/
Combinatorial exploration

Grammar → Explosion

Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term
Term

Adversary of stochastic testing
Explosion examples

Number of generated terms grows fast with depth and eventually explodes (becomes greater than $18446744073709551616$).
Controlled explosion

Grammar

Recursion control

Depth control

\ldots

\text{+ other mechanisms}
Control mechanisms

- depth control
  - intuitive definition

- recursion control
  - nested unfolding of sorts

- equivalence control
  - building equivalence classes
Control mechanisms (contd.)

- balance control
  - limit the preceding levels

- combination control
  - limit Cartesian product
  - pair-wise testing

- context control
  - enforce context conditions
What to test in the XML

- levels of XML file conformance
- levels of XML processor conformance
- grammar features: attributes, references, ...
- advanced features: namespaces, schema-related markup, ...
- secondary features: header, scalability, ...
Design of **Geno**

Grammar-based testing tool generates terms in a combinatorial way.
Work on Geno

Change the input language: grammar adaptation; change the output language: serialisation.
Case study with **Geno**
XML validators

- C# .NET-based
  - wrapper written

- Sun Multi-Schema XML Validator 1.2
  - used as is

- Python XSV
  - wrapper written
Results

- The infrastructure of the XML-based data generator

- The case study: XHTML Strict 1.1

---

- Generation process visualisation

- Illustration and rationalisation of control mechanisms
Scenarios

- Huge valid test data set
- Grammar mutation
- Point-wise stress testing

<table>
<thead>
<tr>
<th></th>
<th>Depth reached</th>
<th>Sorts</th>
<th>Constructors in the signature</th>
<th>Terms total</th>
<th>Terms of the root sort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>8</td>
<td>234</td>
<td>478</td>
<td>9914261</td>
<td>37240</td>
</tr>
<tr>
<td>Mutation</td>
<td>5</td>
<td>234</td>
<td>684</td>
<td>347339</td>
<td>64247</td>
</tr>
<tr>
<td>Stress</td>
<td>1000</td>
<td>5</td>
<td>6</td>
<td>1500</td>
<td>499</td>
</tr>
</tbody>
</table>
BUGS — better say “differences”

- Third outcome: lax validation, warnings, etc
- Duplicate attribute drives C# and Python APIs mad
- Stress testing

---

- FOR cycle
- Running in parallel
Thanks for your attention!