Context Logic and XML update

Gareth Smith

December 12, 2006
<form action="http://www.imperial.ac.uk/script/navhandler.asp" method="post" class="left">
<label for="formbox1"> </label>
<select name="quickNav" id="formbox1" tabindex="2">
<option selected="selected" value="http://www.ic.ac.uk/" >Quick Navigation </option>
<option>-----------------------------</option>
<option>Imperial home page</option>
<option>A-Z of Departments</option>
<option>Courses</option>
<option>Research</option>
<option>Alumni</option>
<option>-----------------------------</option>
<option>Faculty of Engineering</option>
<option>Faculty of Life Sciences</option>
<option>Faculty of Medicine</option>
<option>Faculty of Physical Sciences</option>
<option>Business School</option>
<option>Grad. Sch. Life Sciences &amp; Medicine</option>
<option>-----------------------------</option>
<option>Spectrum (restricted to College users)</option>
<option>People finder</option>
<option>Help</option>
</select>
<input name="nav" type="image" tabindex="3" value="Go!" src="/images/go.gif" alt="Quick navigation go button" align="top" />
</form>
In Practice: DOM

- W3C Standard
- Object Oriented Model
- Written in IDL and English
- Specified as Interfaces.
- Implemented in
  - C++
  - Java
  - Javascript
  - Python
  - Perl
  - . . .
In Theory: Context Logic

- A Generalisation of Separation Logic
- A formal way of describing and reasoning about structured data.
- Particularly Trees.
In Theory: Context Logic

- A Generalisation of Separation Logic
- A formal way of describing and reasoning about structured data.
- Particularly Trees.
- We can describe:

Trees:
- $P$
- $K(P)$
- $K \triangle P$

Contexts:
- $K$
- $P \triangleright P$
The Plan

- Understanding DOM
- Specification of an XML update library
- Study and critique of DOM implementations
- Careful implementation of our specification
- High / low level reasoning to show correctness of our implementation.
Data Model

\[ g ::= T \mid g \oplus g \]  
A grove – a set of trees

\[ T ::= \emptyset \mid \text{tag}_{id}[F]_{id'} \]  
A tree, which may be empty

\[ F ::= T \mid F \otimes F \]  
A forest – a list of trees

\[ CG ::= CT \mid CT \oplus g \]  
A grove context

\[ CT ::= - \mid \text{tag}_{id}[CF]_{id'} \]  
A tree context

\[ CF ::= CT \mid CT \otimes T \mid T \otimes CT \]  
A forest context

\ldots \text{where } \emptyset \text{ is the identity of both } \oplus \text{ and } \otimes, \oplus \text{ is commutative and } \otimes \text{ is not. } ids \text{ are unique. This is enforced in the operational semantics.} \]
Comparable Work

- Peter O’Hearn and the East London Massive
- Matthew Parkinson
- Philippa, Uri, etc
Future Work

- Study and critique of DOM implementations
- Careful implementation of our specification
- High / low level reasoning to show correctness of our implementation.
append(@parent, @newChild) Appends the tree newChild to the tree parent’s forest of children.

insertBefore(@parent, @ref, @newChild) Inserts the tree newChild into the tree parent’s forest of children, before the tree ref.

removeChild(@parent, @child) Removes the tree child from the tree parent’s forest of children. Note: child is not destroyed - simply taken out of the tree containing parent, to be considered separately.

s = getNodeName(@n) s becomes equal to the tag of the tree n.
@f = getChildNodes(@n)  
f changes to point to the forest of the children of the tree n.

@p = getParentNode(@n)  
p changes to point to the parent of n, if it exists, and faults otherwise.

@n = createElement(tag)  
creates a new tree in the grove, with a tag equal to tag. @n changes to point to this new tree.

i = length(@f)  
i becomes equal to the number of trees at the top level of the forest pointed to by f.

@n = item(@f, i)  
n changes to point to the (i + 1)th node in the list which is pointed to by f.
\[
g \equiv (\emptyset \triangleright CG(tag_{parent}[F]_{id}))(tag'_{newChild}[F']_{id'})
g' \equiv CG(tag_{parent}[F \otimes tag'_{newChild}[F']_{id'}]_{id})
\]
\[
\text{append}(\mathbin{@parent, \mathbin{@newChild}), s, g \rightarrow s, g'
\]
\[
g \not\equiv (\emptyset \triangleright CG(tag_{parent}[F]_{id}))(tag'_{newChild}[F']_{fid})
\]
\[
\text{append}(\mathbin{@parent, \mathbin{@newChild}), s, g \rightarrow \text{fault}
\]