Reasoning about Client-side Programming

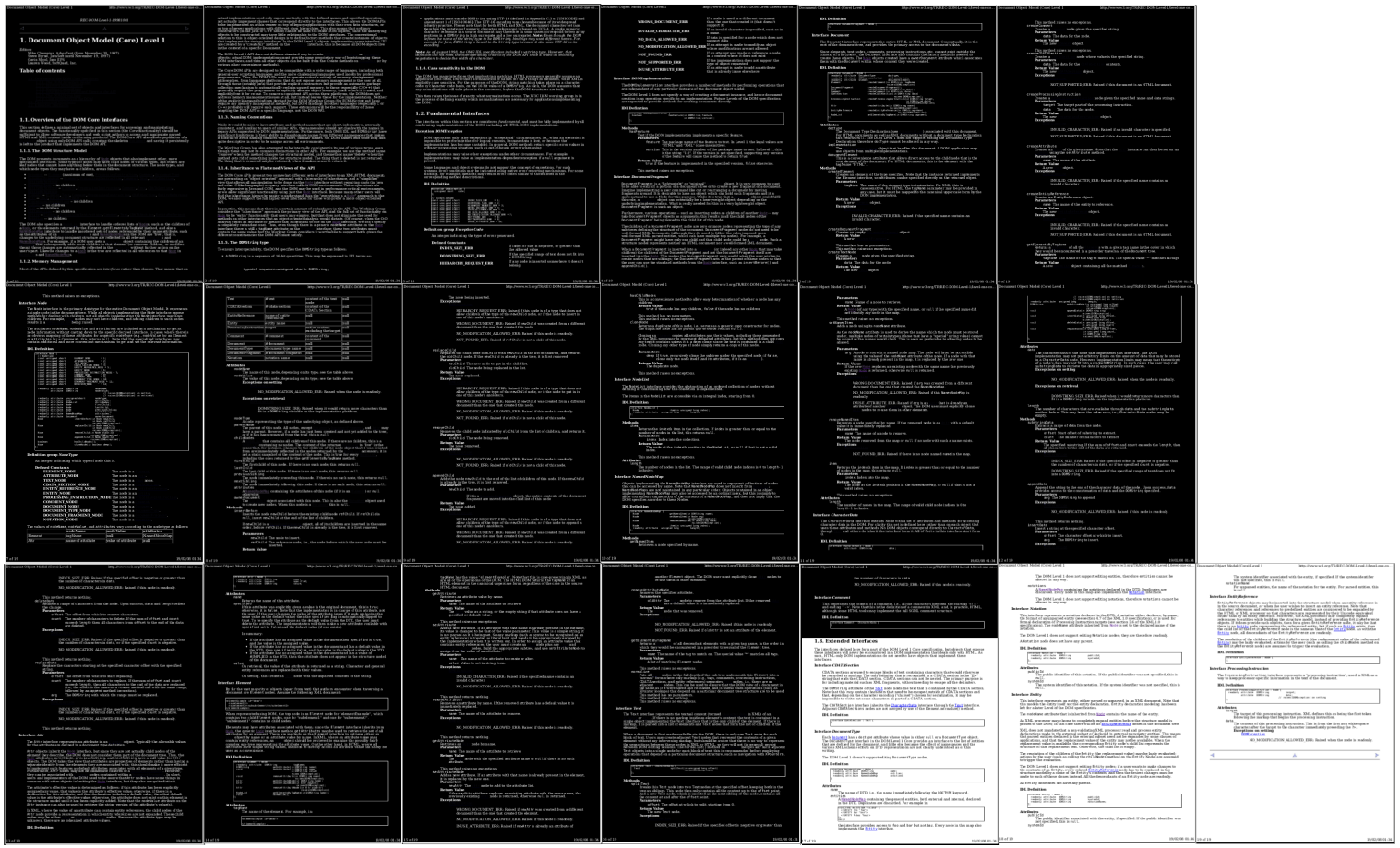
Philippa Gardner, Gareth Smith, Mark Wheelhouse,

Adam Wright and Uri Zarfaty

Imperial College London

Funding: EPSRC, Microsoft and Royal Academy of Engineering.

DOM Core Level 1: Fundamental Interfaces



A Formal Specification of DOM

O'Hearn, Reynolds and Yang introduced local resource reasoning about heap update in their work on Separation Logic. POPL 2001

Calcagno, G. and Zarfaty introduced local resource reasoning about tree update using Context Logic. POPL 2005 and Plotkin's festschrift

G., Smith, Wheelhouse and Zarfaty specified Featherweight DOM using local resource reasoning based on Context Logic. PODS 2008
Unlike DOM, we obtain complete reasoning for straight-line code.

G. and Smith extended the work to DOM Core Level 1: Fundamental Interfaces. Smith's thesis

Example: `getChildNodes` specification

$$\{\mathsf{NAME}_{\mathtt{N}} \lessdot \mathsf{EA{:}EA} \gtrdot^{\mathtt{IRN}}_{\mathtt{TP}}{}_{\mathtt{AIDN}} [\mathsf{F{:}D}]_{\mathtt{FID}} \mathsf{VAL} \wedge \mathtt{kids} \doteq \mathtt{Y}\}$$

$$\mathtt{kids} := \mathtt{getChildNodes}(\mathtt{N})$$

$$\{\mathsf{NAME}_{\mathtt{N\{Y/kids\}}} \lessdot \mathsf{EA{:}EA} \gtrdot^{\mathtt{IRN}}_{\mathtt{TP}}{}_{\mathtt{AIDN}} [\mathsf{F{:}D}]_{\mathtt{FID}} \mathsf{VAL} \wedge \mathtt{kids} \doteq \mathtt{FID}\}$$

Problems Excountered along the Way

`appendChild`

Adds the node newChild at the end of the list of children to this node. If the newChild is already in the tree, it is first removed. ....

Exceptions ...

HIERARCHY_REQUEST_ERR: Raised if this node is of a type that does not allow children of type of the newChild node, or if the node to append is one of this node's ancestors.

This safety error was initially missed by Python mini-DOM, now fixed.

`getElementByTagName`

`normalize`

object-attribute specified

Featherweight DOM

Featherweight DOM Data Structure

trees	t	$::=$	$s_{id}[f]_{fid} \mid \#text_{id}[s]$	identifiers unique
forests	f	$::=$	$\varnothing \mid \langle t \rangle_F \mid f \otimes f$	list of trees
groves	g	$::=$	$\varnothing_G \mid \langle t \rangle_G \mid g \oplus g$	set of trees
strings	s	$::=$	$\varnothing_S \mid c \mid s \cdot s$	list of characters

We omit the subscripts F, G and S, the identifiers and the angle brackets when the meaning is clear.

Example: Witter Data

$$
\begin{aligned}
&\mathrm{witter}_1[\\
&\ \langle \mathrm{update}_3[\mathrm{user}_4[\#\mathrm{text}_5[\text{“username”}]]_6 \otimes \#\mathrm{text}_7[\text{“Hello....”}]]_8\rangle_{\mathsf{F}} \otimes\\
&\ \langle \mathrm{update}_9[\mathrm{user}_{10}[\#\mathrm{text}_{11}[\text{“username”}]]_{12} \otimes \#\mathrm{text}_{13}[\text{“Ciao....”}]]_{14}\rangle_{\mathsf{F}} \otimes\\
&\qquad\vdots\\
&]_2
\end{aligned}
$$

Some angle brackets omitted.

Featherweight DOM

Library commands

$$
\begin{array}{rl}
C ::= & \texttt{appendChild}\ (\texttt{parent}, \texttt{newChild}) \\
| & \texttt{removeChild}\ (\texttt{parent}, \texttt{oldChild}) \\
| & \texttt{name} := \texttt{getNodeName}\ (\texttt{node}) \\
| & \texttt{id} := \texttt{getParentNode}\ (\texttt{node}) \\
| & \texttt{fid} := \texttt{getChildNodes}\ (\texttt{node}) \\
| & \texttt{node} := \texttt{createNode}\ (\texttt{Name}) \\
| & \texttt{node} := \texttt{item}\ (\texttt{list}, \texttt{Int}) \\
| & \texttt{str} := \texttt{substringData}\ (\texttt{node}, \texttt{Offset}, \texttt{Count}) \\
| & \texttt{appendData}\ (\texttt{node}, \texttt{Arg}) \\
| & \texttt{deleteData}\ (\texttt{node}, \texttt{Offset}, \texttt{Count}) \\
| & \texttt{node} := \texttt{createTextNode}\ (\texttt{Str}) \\
\text{Core Language} \quad | & \texttt{id} := \texttt{Id} \mid \texttt{str} := \texttt{Str} \mid \texttt{int} := \texttt{Int} \mid \texttt{bool} := \texttt{Bool} \\
| & \texttt{C ; C} \mid \texttt{if Bool then C else C} \\
| & \texttt{while Bool do C} \mid \texttt{skip} \mid \texttt{local var in C} \mid \ldots
\end{array}
$$

Implementable Commands

```
length := getLength (list)
length := getDataLength (node)
insertBefore (parent, newChild, refNode)
replaceChild (parent, newChild, oldChild)
node' := cloneNode (node, Bool)
bool := hasChildNodes (node)
node := getFirstChild (node)
node := getLastChild (node)
sibling := getPreviousSibling (node)
sibling := getNextSibling (node)
str := getData (node)
setData (node, Str)
insertData (node, Offset, Arg)
replaceData (node, Offset, Count, Arg)
new := splitText (node, Offset)
```

Featherweight DOM is minimal and complete for straight-line code.

Example: Witter Assertions

$$update \triangleq \mathrm{update}\,[\mathrm{user}[\#\mathrm{text}] \otimes \#\mathrm{text}]$$

$$witter \triangleq \mathrm{witter}[\Box_{\otimes}\langle \mathrm{true}_{\mathsf{T}}\rangle_{\mathsf{F}} \Longrightarrow \langle update\rangle_{\mathsf{F}}]$$

Lots of shorthand used.

Example: `getChildNodes` specification

Small Axiom

$$\{\mathtt{NAME}_{\mathtt{node}}[\mathtt{F}]_{\mathtt{FID}} \wedge \mathtt{kids} \doteq \mathtt{Y}\}$$

$$\mathtt{kids} := \mathtt{getChildNodes}(\mathtt{node})$$

$$\{\mathtt{NAME}_{\mathtt{node}[\mathtt{Y}/\mathtt{kids}]}[\mathtt{F}]_{\mathtt{FID}} \wedge \mathtt{kids} \doteq \mathtt{FID}\}$$

Small axioms specify the behaviour of commands on their footprints.

Example: `getChildNodes` specification

Small Axiom

$$\{\texttt{witter}_{\texttt{node}}[\texttt{F}]_{\texttt{FID}} \wedge \texttt{updates} \doteq \texttt{Y}\}$$

$$\texttt{updates} := \texttt{getChildNodes}(\texttt{node})$$

$$\{\texttt{witter}_{\texttt{node}}[\texttt{F}]_{\texttt{FID}} \wedge \texttt{updates} \doteq \texttt{FID}\}$$

Frame Rule and Consequence

$$\left\{ \begin{array}{l} <\texttt{witter}_{\texttt{node}}[\texttt{F}]\texttt{FID}>_{\texttt{G}} \oplus \\ <\texttt{updatesILike}_{\texttt{someUpdates}}[\texttt{F}']_{\texttt{FID}'}>_{\texttt{G}} \wedge \texttt{len}(\texttt{F}) \geq 3 \wedge \texttt{updates} \doteq \texttt{Y} \end{array} \right\}$$

$$\texttt{updates} := \texttt{getChildNodes}(\texttt{node})$$

$$\left\{ \begin{array}{l} <\texttt{witter}_{\texttt{node}}[\texttt{F:F}]\texttt{FID}>_{\texttt{G}} \oplus \\ <\texttt{updatesILike}_{\texttt{someUpdates}}[\texttt{F}']_{\texttt{FID}'}>_{\texttt{G}} \wedge \texttt{len}(\texttt{F}) \geq 3 \wedge \texttt{updates} \doteq \texttt{FID} \end{array} \right\}$$

The frame rule declares that the rest of the data remains unchanged.

Example: Simple Program

```
updates := getChildNodes (node);
thirdkid := item (updates, 2);
appendChild (someUpdates, thirdkid)
```

Example: Simple Program Reasoning

$$\left\{ \begin{array}{l} <\text{witter}_{\texttt{node}}[\mathsf{F}]_{\texttt{FID}} \wedge witter>_{\mathsf{G}} \oplus \\ <\text{updatesILike}_{\texttt{someUpdates}}[\mathsf{F}']_{\texttt{FID}'}>_{\mathsf{G}} \wedge \text{len}(\mathsf{F}) \geq 3 \end{array} \right\}$$

```
updates := getChildNodes(node) ;
```

$$\left\{ \begin{array}{l} <\text{witter}_{\texttt{node}}[\mathsf{F}]_{\texttt{updates}} \wedge witter>_{\mathsf{G}} \oplus \\ <\text{updatesILike}_{\texttt{someUpdates}}[\mathsf{F}']_{\texttt{FID}'}>_{\mathsf{G}} \wedge \text{len}(\mathsf{F}) \geq 3 \end{array} \right\}$$

$$\left\{ \begin{array}{l} <\text{witter}_{\texttt{node}}[\mathsf{F}_1 \otimes <\text{update}_{\texttt{ID}''}[\mathsf{F}'']_{\texttt{FID}''}>_{\mathsf{F}} \otimes \mathsf{F}_2]_{\texttt{updates}} \wedge witter>_{\mathsf{G}} \oplus \\ <\text{updatesILike}_{\texttt{someUpdates}}[\mathsf{F}']_{\texttt{FID}'}>_{\mathsf{G}} \wedge \text{len}(\mathsf{F}_1) = 2 \end{array} \right\}$$

Example: Simple Program Reasoning

$$\left\{ \begin{array}{l} <_{\mathsf{witter}}\mathsf{node}[\mathsf{F}]\mathtt{FID} \wedge witter>_{\mathsf{G}} \oplus \\ <_{\mathsf{updatesILike}}\mathtt{someUpdates}[\mathsf{F}']_{\mathtt{FID}'}>_{\mathsf{G}} \wedge \mathsf{len}(\mathsf{F}) \geq 3 \end{array} \right\}$$

$\mathtt{updates} := \mathtt{getChildNodes}(\mathtt{node})$;

$$\left\{ \begin{array}{l} <_{\mathsf{witter}}\mathsf{node}[\mathsf{F}_1 \otimes <_{\mathsf{update}}{}_{\mathtt{ID}''}[\mathsf{F}'']_{\mathtt{FID}''}>_{\mathsf{F}} \otimes \mathsf{F}_2]\mathtt{updates} \wedge witter>_{\mathsf{G}} \oplus \\ <_{\mathsf{updatesILike}}\mathtt{someUpdates}[\mathsf{F}']_{\mathtt{FID}'}>_{\mathsf{G}} \wedge \mathsf{len}(\mathsf{F}_1) = 2 \end{array} \right\}$$

$\mathtt{thirdChild} := \mathtt{item}(\mathtt{updates}, 2)$;

$$\left\{ \begin{array}{l} <_{\mathsf{witter}}\mathsf{node}[\mathsf{F}_1 \otimes <_{\mathsf{update}}\mathtt{thirdChild}[\mathsf{F}'']_{\mathtt{FID}''}>_{\mathsf{F}} \otimes \mathsf{F}_2]\mathtt{updates} \wedge witter>_{\mathsf{G}} \oplus \\ <_{\mathsf{updatesILike}}\mathtt{someUpdates}[\mathsf{F}']_{\mathtt{FID}'}>_{\mathsf{G}} \wedge \mathsf{len}(\mathsf{F}_1) = 2 \end{array} \right\}$$

$\mathtt{appendChild}(\mathtt{someUpdates}, \mathtt{thirdChild})$

$$\left\{ \begin{array}{l} <_{\mathsf{witter}}\mathsf{node}[\mathsf{F}_1 \otimes \mathsf{F}_2]\mathtt{updates} \wedge witter>_{\mathsf{G}} \oplus \\ <_{\mathsf{updatesILike}}\mathtt{someUpdates}[\mathsf{F}' \otimes <_{\mathsf{update}}\mathtt{thirdChild}[\mathsf{F}'']_{\mathtt{FID}''}>_{\mathsf{F}}]_{\mathtt{FID}'}>_{\mathsf{G}} \end{array} \right\}$$

Example: Witter Mashup

The Mashup client

$$\mathbf{MC}(\mathsf{clientPage}) = \left(\begin{array}{l} \mathsf{html}_1[\\ \quad \vdots \\ \quad \mathsf{updatesILike}_n[\varnothing]_m \ , \ \mathbf{CODE} \\ \quad \vdots \\ ]_2 \end{array} \right)$$

Example: Witter Mashup

CODE =

```
document = fetchDocument("clientPage");
updatesDoc = fetchDocument("witter.example.com/updates");
runScript("spamcheck.example.com");
someUpdates = ...
updates = getChildNodes(updatesDoc)
length := getLength(updates); i = 0;
while (i < length) {
  update = item(updates, i);
  i = i + 1;
  userNode = getFirstChild(update):
  userName = getNodeValue(userNode);
  if (isUserSpammer(userName) == false) {
    appendChild(someUpdates, update);   }   else { skip; }   }
```

Example: Witter Mashup

The Witter updates list $\mathbf{MC}(\text{witter.example.com/updates}) =$

$$\left(\begin{array}{l}
\text{witter}_1[\\
\quad \text{update}_3[\text{user}_4[\#\text{text}_5[\text{“username”}]]_6 \otimes \#\text{text}_7[\text{“Update...”}]]_8 \otimes \\
\quad \text{update}_9[\text{user}_{10}[\#\text{text}_{11}[\text{“username”}]]_{12} \otimes \#\text{text}_{13}[\text{“Update...”}]]_{14} \otimes \\
\qquad \vdots \\
]_2, \\
\texttt{skip}
\end{array}\right)$$

The Spam Checking service $\mathbf{MC}(\text{spamcheck.example.com}) =$

$$(\ldots, \texttt{function isUserSpammer(username)}\ \{\ldots\})$$

Example: Witter Mashup

CODE =

```
document = fetchDocument("clientPage");
updatesDoc = fetchDocument("witter.example.com/updates");
runScript("spamcheck.example.com");
someUpdates = ...
updates = getChildNodes(updatesDoc)
length := getLength(updates); i = 0;
while (i < length) {
  update = item(updates, i);
  i = i + 1;
  userNode = getFirstChild(update):
  userName = getNodeValue(userNode);
  if (isUserSpammer(userName) == false) {
    appendChild(someUpdates, update);   }   else { skip; }   }
```

Example: Witter Mashup Reasoning

The Witter component

$$\mathbf{MCspec}(\text{witter.example.com}) = (witter, \varnothing_{\mathsf{G}}, \varnothing_{\mathsf{G}}, \{\}, \{\})$$

The Spamcheck component

$$\mathbf{MCSpec}(\text{spamcheck.example.com}) =$$

$$(true, \varnothing_{\mathsf{G}}, \gamma(\texttt{isUserSpammer}), \{\}, \{\texttt{isUserSpammer} : spam\})$$

$$spam \triangleq (\texttt{param} = \textsc{String}, \texttt{ret} = true \vee \texttt{ret} = false, \{\})$$

The client code $\mathbf{MCspec}(\text{clientPage}) =$

$$(htmlSpec, mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \varnothing_{\mathsf{G}}, true, \texttt{Mods}, \{\})$$

$$htmlSpec \triangleq \text{html}[\Diamond\text{updatesILike}_n[\varnothing_{\mathsf{F}}]]$$

Example: Witter Mashup Reasoning

$\{mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \varnothing_{\mathsf{G}}\}$

```
document = fetchDocument("clientPage");
```

$\{mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \langle \text{html}\texttt{document}[\Diamond \text{updatesILike}_n[\varnothing_{\mathsf{F}}]]\rangle_{\mathsf{G}}\}$

```
updatesDoc = fetchDocument("witter.example.com/updates");
```

$$\left\{\begin{array}{l} mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \\ \langle \text{html}\texttt{document}[\Diamond \text{updatesILike}_n[\varnothing_{\mathsf{F}}]] \otimes \text{witter}\texttt{updatesDoc}[\Box_{\otimes}\langle \text{true}_{\mathsf{T}}\rangle_{\mathsf{F}} \implies \langle update\rangle_{\mathsf{F}}]\rangle_{\mathsf{G}} \end{array}\right\}$$

```
runScript("spamcheck.example.com");
```

$$\left\{\begin{array}{l} mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \gamma(\texttt{userIsSpammer}) \wedge \\ \langle \text{html}\texttt{document}[\Diamond \text{updatesILike}_n[\varnothing_{\mathsf{F}}]] \otimes \text{witter}\texttt{updatesDoc}[\Box_{\otimes}\langle \text{true}_{\mathsf{T}}\rangle_{\mathsf{F}} \implies \langle update\rangle_{\mathsf{F}}]\rangle_{\mathsf{G}} \end{array}\right\}$$

```
someUpdates = ...
```

$$\left\{\begin{array}{l} mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \gamma(\texttt{userIsSpammer}) \wedge \\ \langle \text{html}\texttt{document}[\Diamond \text{updatesILike}\texttt{someUpdates}[\varnothing_{\mathsf{F}}]] \otimes \text{witter}\texttt{updatesDoc}[\Box_{\otimes}\langle \text{true}_{\mathsf{T}}\rangle_{\mathsf{F}} \implies \langle update\rangle_{\mathsf{F}}]\rangle_{\mathsf{G}} \end{array}\right\}$$

```
updates = getChildNodes(updatesDoc)
length := getLength(updates); i = 0;
```

$$\left\{\begin{array}{l} mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \gamma(\texttt{userIsSpammer}) \wedge \\ \langle \text{html}\texttt{document}[\Diamond \text{updatesILike}\texttt{someUpdates}[\Box_{\otimes}\langle \text{true}_{\mathsf{T}}\rangle_{\mathsf{F}} \implies \langle update\rangle_{\mathsf{F}}]] \otimes \\ \text{witter}\texttt{updatesDoc}[\Box_{\otimes}\langle \text{true}_{\mathsf{T}}\rangle_{\mathsf{F}} \implies \langle update\rangle_{\mathsf{F}}]\rangle_{\mathsf{G}} \end{array}\right\}$$

```
while (i < length) {
  update = item(updates, i);
```

$$\left\{ \begin{array}{l} mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \gamma(\texttt{userIsSpammer}) \wedge \\ \langle \text{html}\texttt{document}[\Diamond \text{updatesILike}\texttt{someUpdates}[\Box_{\otimes} \langle \text{true}_{\mathsf{T}} \rangle_{\mathsf{F}} \implies \langle update \rangle_{\mathsf{F}}]] \otimes \\ \text{witter}\texttt{updatesDoc}[\Box_{\otimes}(\langle \text{true}_{\mathsf{T}} \rangle_{\mathsf{F}} \implies \langle update \rangle_{\mathsf{F}}) \wedge \\ Update2010(F_1 \otimes \text{update}\texttt{update}[\text{user}[\#\text{text}[X]] \otimes \#\text{text}] \otimes F_2 \wedge \text{len}(F_1) = \texttt{i})]\rangle_{\mathsf{G}} \end{array} \right\}$$

```
  i = i + 1;
  userNode = getFirstChild(update):
  userName = getNodeValue(userNode);
```

$$\left\{ \begin{array}{l} mc(\text{witter.example.com}) \wedge mc(\text{spamcheck.example.com}) \wedge \gamma(\texttt{userIsSpammer}) \wedge \\ \langle \text{html}\texttt{document}[\Diamond \text{updatesILike}\texttt{someUpdates}[\Box_{\otimes} \langle \text{true}_{\mathsf{T}} \rangle_{\mathsf{F}} \implies \langle update \rangle_{\mathsf{F}}]] \otimes \\ \text{witter}\texttt{updatesDoc}[\Box_{\otimes}(\langle \text{true}_{\mathsf{T}} \rangle_{\mathsf{F}} \implies \langle update \rangle_{\mathsf{F}}) \wedge (F_1 \otimes \text{update}\texttt{update}[\\ \quad \text{user}\texttt{userNode}[\#\text{text}[X \wedge X = \texttt{userName}]] \otimes \#\text{text} \\ ] \otimes F_2 \wedge \text{len}(F_1) = \texttt{i} - 1)]\rangle_{\mathsf{G}} \end{array} \right\}$$

```
  if (isUserSpammer(userName) == false) {
    appendChild(someUpdates, update);
  }
  else { skip; }
}
```

Justification of Correctness

Soundness Result

Featherweight DOM axiomatic semantics is correct with respect to the Featherweight DOM operational semantics. PODS 2008

DOM Core Level 1 operational semantics done, Smith's thesis.

We will not prove soundness this way.

Future Soundness Results

Implementations of DOM Core Level 1 will be tested using its axiomatic semantics.

Other Work

The Fiction of Separation, Dinsdale-Young, G. and Wheelhouse.

DOM Core Level 1 operational semantics Gareth's thesis, JavaScript operational semantics/secure Javascript fragments, Taly, Maffeis and Mitchell, secure JavaScript/DOM fragments, future.

Reasoning about File Systems, G. and Wright.

Abstract Specification of Concurrent Modules, Dinsdale-Young, Dodds, G., Parkinson and Vafeiadis.

Reasoning about Concurrent XML Update, G. and Wheelhouse.