Pop Quiz!
Question 1

Assume d is a Document object.

var e = d.createElement("div");
Question 1

Assume $d$ is a Document object.

```javascript
var e = d.createElement("div");
```

Assume $d$ and $e$ remain unchanged.
Question 1

Assume d is a Document object.

var e = d.createElement("div");

Assume d and e remain unchanged.

Is it guaranteed that e.ownerDocument == d is always true?

a) Yes
b) No
Question 1

Assume d is a Document object.

var e = d.createElement("div");

Assume d and e remain unchanged.

Is it guaranteed that e.ownerDocument == d is always true?

b) No
Question 2

Which of the following can a script do to cause the browser to run (or re-run) some other script?

a) Remove a script node from a document and insert it somewhere else.
b) Replace a child text node of a script node.
c) Assign a new value to an already-present src attribute of a script node.
d) All of the above.
e) None of the above.
Question 2

Which of the following can a script do to cause the browser to run (or re-run) some other script?

a) Remove a script node from a document and insert it somewhere else.
Question 2

Which of the following can a script do to cause the browser to run (or re-run) some other script?

a) Remove a script node from a document and insert it somewhere else.

b) Replace a child text node of a script node.
Question 2

Which of the following can a script do to cause the browser to run (or re-run) some other script?

a) Remove a script node from a document and insert it somewhere else.

b) Replace a child text node of a script node.

c) Assign a new value to an already-present src attribute of a script node.

d) All of the above.

e) None of the above.
Question 2

Which of the following can a script do to cause the browser to run (or re-run) some other script?

a) Remove a script node from a document and insert it somewhere else.
b) Replace a child text node of a script node.
c) Assign a new value to an already-present src attribute of a script node.
d) All of the above.
Question 2

Which of the following can a script do to cause the browser to run (or re-run) some other script?

a) Remove a script node from a document and insert it somewhere else.

b) Replace a child text node of a script node.

c) Assign a new value to an already-present src attribute of a script node.

d) All of the above.

e) None of the above.
Question 2

Which of the following can a script do to cause the browser to run (or re-run) some other script?

a) Remove a script node from a document and insert it somewhere else.
b) Replace a child text node of a script node.
c) Assign a new value to an already-present src attribute of a script node.
d) All of the above.
e) None of the above.

e) None of the above.
Question 3

A handler for a button click can always get a reference to the window in which the user clicked.

a) True
b) False
Question 3

A handler for a button click can always get a reference to the window in which the user clicked.

a) True

True. The handler can just use the expression self (or window).
Question 3

A handler for a button click can always get a reference to the window in which the user clicked.

b) False

No, false. `self` is statically scoped to refer to the window where the code is defined.
Question 3

A handler for a button click can always get a reference to the window in which the user clicked.

a) True

No, true. Button handlers can always check the `ownerDocument` property of the button node.
Question 3

A handler for a button click can always get a reference to the window in which the user clicked.

b) False

No, false. If a different handler runs first, it may move the button node to a different window!
Web Script Semantics

Web script semantics are a bit peculiar.
Web Script Semantics

Web script semantics are a bit peculiar.

- Web scripts manipulate interconnected browser structures.
Web Script Semantics

Web script semantics are a bit peculiar.

- Web scripts manipulate interconnected browser structures.
- Web scripts are event-driven (user input, network responses, timer events, etc.).
Web Script Semantics

Web script semantics are a bit peculiar.

- Web scripts manipulate interconnected browser structures.
- Web scripts are event-driven (user input, network responses, timer events, etc.).
- Web scripts have interesting language constructs (first-class functions, dynamic evaluation, self, etc.).
Why Formalize This Stuff?

- We want to perform a rigorous study of browser information security policies.
Why Formalize This Stuff?

- We want to perform a rigorous study of browser information security policies.
- This demands a rigorous definition of browser behavior.
Simplifying Assumptions

- Abstract away from some lower-level details (parsing, rendering, DNS).
Simplifying Assumptions

- Abstract away from some lower-level details (parsing, rendering, DNS).
- Make the semantics deterministic, modulo the order of input events.
Simplifying Assumptions

- Abstract away from some lower-level details (parsing, rendering, DNS).
- Make the semantics deterministic, modulo the order of input events.
- Model the BOM operations semantics but not the details of the JavaScript language.
Simplifying Assumptions

- Abstract away from some lower-level details (parsing, rendering, DNS).
- Make the semantics deterministic, modulo the order of input events.
- Model the BOM operations semantics but not the details of the JavaScript language.
- Omit all security mechanisms.
Formalization Overview

We’ve designed a formal web browser semantics that . . .

▶ includes many key browser features.
We’ve designed a formal web browser semantics that . . .

- includes many key browser features.
- operates in a small-step style.
Formalization Overview

We’ve designed a formal web browser semantics that . . .

▶ includes many key browser features.
▶ operates in a small-step style.
▶ is declarative (in the style of logical inference rules).
We’ve designed a formal web browser semantics that . . .

- includes many key browser features.
- operates in a small-step style.
- is declarative (in the style of logical inference rules).
- is written down in a strongly-typed programming language (OCaml).
Included Features

- Multiple windows and pages
- Mutable document node trees
- Buttons and text boxes with handlers
- Network requests and responses with cookies
- Scripts with first-class functions, eval, and AJAX requests
Omitted Features

- Browsing history
- HTTP error codes and redirects
- “timeout” events in scripts
- javascript: URLs
- file: URLs
Related Work
Whole Browser Formalizations

- HTML5
Whole Browser Formalizations

- HTML5
Whole Browser Formalizations

- HTML5
Other Formalizations

Formalization Details
Reactive Systems

![Diagram showing consumer and producer states]

- Consumer States
- Producer States
Reactive Systems

Consumer States ← i → Producer States
Reactive Systems
Web Browser Consumer State

- Window store
- Page store
- Node store
- Activation record store
- Cookie store
- List of open network connections
Web Browser Producer State

- Window store
- Page store
- Node store
- Activation record store
- Cookie store
- List of open network connections
- Task list
**window:**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>string (optional)</td>
</tr>
<tr>
<td>opener</td>
<td>reference to a window (optional)</td>
</tr>
<tr>
<td>current page</td>
<td>reference to a page</td>
</tr>
</tbody>
</table>
# Page Store

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>page:</td>
<td>---</td>
</tr>
<tr>
<td>address</td>
<td>URL</td>
</tr>
<tr>
<td>root node</td>
<td>reference to a node</td>
</tr>
<tr>
<td>environment</td>
<td>reference to an activation record</td>
</tr>
<tr>
<td>script queue</td>
<td>list of scripts or placeholders</td>
</tr>
</tbody>
</table>
Network Connection List

network connection:

- connection for document request: URL, reference to a window
- connection for script request: URL, reference to a node
- connection for AJAX request: URL, reference to a page, expression
Selected Inputs

From the user:
- `load_in_new_window(url)`
- `click_button(win, n)`

From the network:
- `receive(d, n, resp)`
Selected Outputs

To the user:

- win_closed(win)
- page_updated(win, doc)

To the network:

- send(d, req_uri, cookies, msg)
What’s Next?
Using Our Browser Semantics

- Primarily, our formalization should be viewed as a human-readable template.
Using Our Browser Semantics

- Primarily, our formalization should be viewed as a human-readable template.
- Others may be interested in slightly different features.
Using Our Browser Semantics

- Primarily, our formalization should be viewed as a human-readable template.
- Others may be interested in slightly different features.
- The semantics may need to be translated to a different machine-consumable form.
Work in Progress

- Translate browser formalization into Coq.

- Define security policies for the browser in terms of “reactive noninterference” (Bohannon, et al., CCS 2009).

- Prove the soundness of some enforcement mechanisms for these policies.

- Gain a better understanding of end-to-end web browser security.
Work in Progress

- Translate browser formalization into Coq.
- Define security policies for the browser in terms of “reactive noninterference” (Bohannon, et al., CCS 2009). 


Work in Progress

- Translate browser formalization into Coq.
- Define security policies for the browser in terms of “reactive noninterference” (Bohannon, et al., CCS 2009).
- Prove the soundness of some enforcement mechanisms for these policies.
Work in Progress

- Translate browser formalization into Coq.
- Define security policies for the browser in terms of “reactive noninterference” (Bohannon, et al., CCS 2009).
- Prove the soundness of some enforcement mechanisms for these policies.
- Gain a better understanding of end-to-end web browser security.
Thank You