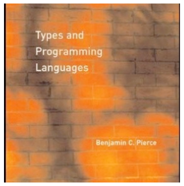
The Essence of JavaScript (Redux)

Guha, Saftoiu, and Krishnamurthi
Brown University



http://www.cs.brown.edu/research/plt/dl/jssem/v1/

Why another JavaScripts Semantics

- Need simple compact account of JS suited for formal reasoning
- Unambiguous account of the core semantics of the language
- Simplify implementations by getting rid of premature optimizations
- But…
  - How do we know we have a semantics of *JavaScript*?
  - How *simple* should it be? What core features to include?
  - How to handle all the builtins and primitives?

Why another lecture of JS semantics

- Revisit the paper in the light of
  - our experience encoding objects in the lambda calculus
  - your experience implementing JavaScript

Small-step Contextual Semantics

- Core reduction relation

$$\hookrightarrow$$

Small-step Contextual Semantics

- Expressions and values

$$c = num \mid str \mid bool \mid \textbf{undefined} \mid \textbf{null}$$
$$v = c \mid \textbf{func}(x \cdots)\ \texttt{\{}\ \textbf{return}\ e\ \texttt{\}} \mid \texttt{\{}\ str\texttt{:}v \cdots\ \texttt{\}}$$
$$e = x \mid v \mid \textbf{let}\ \texttt{(}x\ \texttt{=}\ e\texttt{)}\ e \mid e\texttt{(}e\cdots\texttt{)} \mid e\texttt{[}e\texttt{]} \mid e\texttt{[}e\texttt{]}\ \texttt{=}\ e$$
$$\textbf{delete}\ e\texttt{[}e\texttt{]}$$

Small-step Contextual Semantics

- Contexts

$$
\begin{aligned}
E = \bullet \mid\ & \textbf{let}\ \texttt{(}x\ \texttt{=}\ E\texttt{)}\ e \mid E\texttt{(}e \cdots\texttt{)} \mid v\texttt{(}v \cdots\ E\texttt{,}\ e \cdots\texttt{)} \\
\mid\ & \texttt{\{}str\texttt{:}\ v \cdots\ str\texttt{:}E\texttt{,}\ str\texttt{:}e \cdots\ \texttt{\}} \mid E\texttt{[}e\texttt{]} \mid v\texttt{[}E\texttt{]} \mid \\
\mid\ & v\texttt{[}v\texttt{]}\ \texttt{=}\ E \mid \textbf{delete}\ E\texttt{[}e\texttt{]} \mid \textbf{delete}\ v\texttt{[}E\texttt{]} \\
\mid\ & E\texttt{[}e\texttt{]}\ \texttt{=}\ e \mid v\texttt{[}E\texttt{]}\ \texttt{=}\ e
\end{aligned}
$$

Small-step Contextual Semantics

- Reduction

$$\textbf{let}\ (x = v)\ e \hookrightarrow e[x/v] \cdots$$

$$(\textbf{func}(x_1 \cdots x_n)\ \{\ \textbf{return}\ e\ \})(v_1 \cdots v_n) \hookrightarrow e[x_1/v_1 \cdots x_n/v_n]$$

$$\{\ \cdots str{:}\ v \cdots\ \}[str] \hookrightarrow \mathrm{v}$$

$$\frac{str_x \notin (str_1 \cdots str_n)}{\{\ str_1{:}\ v_1\ \ \cdots\ \ str_n{:}\ v_n\ \}\ [str_x] \hookrightarrow \textbf{undefined}}$$

Small-step Contextual Semantics

- Reduction

$$\textbf{delete}\ \texttt{\{}\ str_1\texttt{:}\ v_1 \cdots\ str_i\texttt{:}\ v_x\ \cdots str_x\texttt{:}\ v_n\ \texttt{\}}\ \texttt{[}str_x\texttt{]}$$
$$\hookrightarrow \texttt{\{}\ str_1\texttt{:}\ v_1 \cdots\ str_i\texttt{:}\ v\ \cdots str_n\texttt{:}\ v_n\ \texttt{\}}$$

$$\frac{str_x \notin (str_1 \cdots)}{\textbf{delete}\ \texttt{\{}\ str_1\texttt{:}\ v_1 \cdots\ \texttt{\}}\ \texttt{[}str_x\texttt{]} \hookrightarrow \texttt{\{}\ str_1\texttt{:}\ v_1 \cdots\ \texttt{\}}}$$

References

- Two-level reduction relation

$$\longrightarrow$$

References

- Extending the language with references

$$
\begin{aligned}
l &= \cdots \\
v &= \cdots \mid l \\
\sigma &= (l, v) \cdots \\
e &= \cdots \mid e \;\textsf{\textbf{=}}\; e \mid \textsf{\textbf{ref}}\; e \mid \textsf{\textbf{deref}}\; e \\
E &= \cdots \mid E \;\textsf{\textbf{=}}\; e \mid v \;\textsf{\textbf{=}}\; E \mid \textsf{\textbf{ref}}\; E \mid \textsf{\textbf{deref}}\; E
\end{aligned}
$$

References

- Reduction

$$\frac{e_1 \hookrightarrow e_2}{\sigma E\langle e_1 \rangle \rightarrow \sigma E\langle e_2 \rangle}$$

References

- Reduction

$$\frac{l \notin dom(\sigma) \qquad \sigma' = \sigma, (l, v)}{\sigma E\langle \textbf{ref}\ \ v\rangle \to \sigma' E\langle \mathsf{l}\rangle}$$

$$\sigma E\langle \textbf{deref}\ \ l\rangle \to \sigma E\langle \sigma(l)\rangle$$

$$\sigma E\langle l\ \ \textbf{=}\ \ v\rangle \to \sigma[l/v] E\langle l\rangle$$

Prototypes

$$\frac{str_x \notin (str_1 \cdots str_n) \qquad \texttt{"\_\_proto\_\_"} \notin (str_1 \cdots str_n)}{\texttt{\{}\ str_1\ \texttt{:}\ v_1\ \texttt{,}\ \cdots\ \texttt{,}\ str_n\ \texttt{:}\ v_n\ \texttt{\}}\ \texttt{[}str_x\texttt{]} \hookrightarrow \textbf{undefined}}$$

$$\frac{str_x \notin (str_1 \cdots str_n)}{str_1\ \texttt{:}\ v_1 \cdots \texttt{"\_\_proto\_\_":}\ \textbf{null}\ \cdots\ str_n\ \texttt{:}\ v_n\ \texttt{\}}\ \texttt{[}str_x\texttt{]} \hookrightarrow \textbf{undefined}}$$

$$\frac{str_x \notin (str_1 \cdots str_n) \qquad p = \textbf{ref}\ l}{\texttt{\{}\ str_1\ \texttt{:}\ v_1 \cdots \texttt{"\_\_proto\_\_":}\ p\ \cdots\ str_n\ \texttt{:}\ v_n\ \texttt{\}}\ \texttt{[}str_x\texttt{]} \hookrightarrow (\textbf{deref}\ p)\texttt{[}str_x\texttt{]}}$$

Desugaring Objects

$$[\![e]\!]$$

Desugaring Objects

$desugar[\![\{prop\colon\ e \cdots\}\ ]\!] =$

ref {
  prop : $desugar[\![e]\!] \cdots$,
  "__proto__": (**deref** Object) ["prototype"]
}

Desugaring Objects

$$desugar[\![\texttt{function(}x\cdots\texttt{) \{ } stmt\cdots \texttt{ \} }]\!] =$$

ref {
"code": **func**(this, $x\cdots$) { **return** $desugar[\![stmt\cdots]\!]$ },
"prototype": **ref** { "__proto__": (**deref** Object)["prototype"] } }

Desugaring Objects

$desugar[\![\texttt{new}\ e_f(e \cdots)\}]\!] =$

let (constr = **deref** $desugar[\![e_f]\!]$)
  let (obj = **ref** { "__proto__" : constr["prototype"]})
    constr["code"](obj, $desugar[\![e]\!] \cdots$);
    obj

Desugaring Objects

$desugar[\![obj\,[field]\,(e \cdots)]\!] =$

let (obj = $desugar[\![obj]\!]$)
  let (f = (**deref** obj) [field])
    f["code"] (obj, $desugar[\![e]\!] \cdots$)

$desugar[\![e_f\,(e \cdots)]\!] =$

let (obj = $desugar[\![e_f]\!]$)
  let (f = **deref** obj)
    f["code"] (window, $desugar[\![e]\!] \cdots$)

Desugaring Objects

$desugar[\![obj \ \texttt{instanceof} \ constr]\!] =$

```
let (obj = ref (deref desugar[[obj]]),
     constr = deref desugar[[constr]])
  done: {
    while (deref obj !== null) {
      if ((deref obj)["__proto__"] === constr["prototype"]) {
        break done true }
      else { obj = (deref obj)["__proto__"] } };
    false }
```

Desugaring Objects

$desugar[\![$`this`$]\!] =$ `this` (an ordinary identifier, bound by fun

$desugar[\![e.x]\!] = desugar[\![e]\!]$`["x"]`

Control Flow

$$
\begin{aligned}
label &= \text{(Labels)}\\
e &= \cdots \mid \textbf{if}\ \texttt{(}e\texttt{)}\ \texttt{\{}\ e\ \texttt{\}}\ \textbf{else}\ \texttt{\{}\ e\ \texttt{\}} \mid e\texttt{;}e \mid \textbf{while}\texttt{(}e\texttt{)}\ \texttt{\{}\ e\ \texttt{\}} \mid label\texttt{:\{}\ e\ \texttt{\}}\\
&\mid \textbf{break}\ label\ e \mid \textbf{try}\ \texttt{\{}\ e\ \texttt{\}}\ \textbf{catch}\ \texttt{(}x\texttt{)}\ \texttt{\{}\ e\ \texttt{\}} \mid \textbf{try}\ \texttt{\{}\ e\ \texttt{\}}\ \textbf{finally}\ \texttt{\{}\ e\ \texttt{\}}\\
&\mid \textbf{err}\ v \mid \textbf{throw}\ e\\
E &= \cdots \mid \textbf{if}\ \texttt{(}E\texttt{)}\ \texttt{\{}\ e\ \texttt{\}}\ \textbf{else}\ \texttt{\{}\ e\ \texttt{\}} \mid E\texttt{;}e \mid label\texttt{:\{}\ E\ \texttt{\}}\\
&\mid \textbf{try}\ \texttt{\{}\ E\ \texttt{\}}\ \textbf{catch}\ \texttt{(}x\texttt{)}\ \texttt{\{}\ e\ \texttt{\}} \mid \textbf{try}\ \texttt{\{}\ E\ \texttt{\}}\ \textbf{finally}\ \texttt{\{}\ e\ \texttt{\}} \mid \textbf{throw}\ E\\
E' &= \bullet \mid \textbf{let}\ \texttt{(}x\ \texttt{=}\ v\cdots\ x\ \texttt{=}\ E'\texttt{,}\ x\ \texttt{=}\ e\cdots\texttt{)}\ e \mid E'\texttt{(}e\cdots\texttt{)} \mid v\texttt{(}v\cdots\ E'\texttt{,}\ e\cdots\texttt{)}\\
&\mid \textbf{if}\ \texttt{(}E'\texttt{)}\ \texttt{\{}\ e\ \texttt{\}}\ \textbf{else}\ \texttt{\{}\ e\ \texttt{\}} \mid \texttt{\{}\ str\texttt{:}\ v\cdots\ str\texttt{:}\ E'\texttt{,}\ str\texttt{:}\ e\cdots\ \texttt{\}}\\
&\mid E'\texttt{[}e\texttt{]} \mid v\texttt{[}E'\texttt{]} \mid E'\texttt{[}e\texttt{]}\ \texttt{=}\ e \mid v\texttt{[}E'\texttt{]}\ \texttt{=}\ e \mid v\texttt{[}v\texttt{]}\ \texttt{=}\ E' \mid E'\ \texttt{=}\ e \mid v\ \texttt{=}\ E'\\
&\mid \textbf{delete}\ E'\texttt{[}e\texttt{]} \mid \textbf{delete}\ v\texttt{[}E'\texttt{]} \mid \textbf{ref}\ E' \mid \textbf{deref}\ E' \mid E'\texttt{;}\ e \mid \textbf{throw}\ E'\\
F &= E' \mid label\texttt{:\{}\ F\ \texttt{\}} \quad \text{(Exception Contexts)}\\
G &= E' \mid \textbf{try}\ \texttt{\{}\ G\ \texttt{\}}\ \textbf{catch}\ \texttt{(}x\texttt{)}\ \texttt{\{}\ e\ \texttt{\}} \quad \text{(Local Jump Contexts)}
\end{aligned}
$$

Control Flow

$$\textbf{if } (\textbf{true}) \; \texttt{\{} \; e_1 \; \texttt{\}} \; \textbf{else} \; \texttt{\{} \; e_2 \; \texttt{\}} \hookrightarrow e_1$$

$$\textbf{if } (\textbf{false}) \; \texttt{\{} \; e_1 \; \texttt{\}} \; \textbf{else} \; \texttt{\{} \; e_2 \; \texttt{\}} \hookrightarrow e_2$$

$$v\,\texttt{;}\,e \;\hookrightarrow\; e$$

Control Flow

$$\textbf{while}(e_1)\ \{\ e_2\ \} \hookrightarrow$$

$$\textbf{if}\ (e_1)\ \{\ e_2;\ \textbf{while}(e_1)\ \{\ e_2\ \}\ \}\ \textbf{else}\ \{\ \textbf{undefined}\ \}$$

$$\textbf{throw}\ \ v \hookrightarrow \textbf{err}\ \ v$$

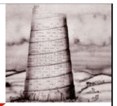
Control Flow

$$\textbf{try } \texttt{\{ } F\langle \textbf{err } v\rangle \texttt{ \} } \textbf{catch } \texttt{(}x\texttt{) \{ } e \texttt{ \}} \hookrightarrow e[x/v]$$

$$\sigma F\langle \textbf{err } v\rangle \rightarrow \sigma \textbf{err } v$$

$$\textbf{try } \texttt{\{ } F\langle \textbf{err } v\rangle \texttt{ \} } \textbf{finally } \texttt{\{ } e \texttt{ \}} \hookrightarrow e\texttt{; } \textbf{err } v$$

Control Flow



$$\textbf{try}\ \texttt{\{}\ G\langle \textbf{break}\ \ label\ \ v\rangle\ \texttt{\}}\ \textbf{finally}\ \texttt{\{}\ e\ \texttt{\}} \hookrightarrow e\texttt{;}\ \ \textbf{break}\ \ label\ \ v$$

$$\textbf{try}\ \texttt{\{}\ v\ \texttt{\}}\ \textbf{finally}\ \texttt{\{}e\ \texttt{\}} \hookrightarrow e\texttt{;}\ \ v$$

$$\mathsf{label}\texttt{:\{}\ G\langle \textbf{break}\ \ \mathsf{label}\ \ v\rangle\ \texttt{\}} \hookrightarrow v$$

$$\frac{label_1 \neq label_2}{label_1\texttt{:\{}\ G\langle \textbf{break}\ \ label_2\ \ v\rangle\ \texttt{\}} \hookrightarrow \textbf{break}\ \ v}$$

$$label\texttt{:}\ \ \texttt{\{}v\texttt{\}} \hookrightarrow v$$

Local Variables

A local variable declaration, `var x = e`, is desugared to an assignment, `x = e`. Furthermore, we add a let-binding at the top of the enclosing function:

let (x = **ref undefined**) $\cdots$

Globals

Global Variables Global variables are subtle. Global variables are properties of the global scope object (`window`), which has a field that references itself:

```
window.window === window ↠ true
```

Therefore, a program can obtain a reference to the global scope object by simply referencing `window`.[4]

As a consequence, globals seem to break lexical scope, since we can observe that they are properties of `window`:

```
var x = 0;
window.x = 50;
x ↠ 50
x = 100;
window.x ↠ 100
```

However, `window` is the only scope object that is directly accessible to JavaScript programs [6, Section 10.1.6]. We maintain lexical scope by abandoning global variables. That is, we simply desugar the obtuse code above to the following:

```
window.x = 0;
window.x = 50;
window.x ↠ 50
window.x = 100;
window.x ↠ 100
```

Although global variables observably manipulate `window`, local variables are still lexically scoped. We can thus reason about local variables using substitution, α-renaming, and other standard techniques.

With

With Statements The `with` statement is a widely-acknowledged JavaScript wart. A `with` statement adds an arbitrary object to the front of the scope chain:

```
function(x, obj) {
  with(obj) {
    x = 50; // if obj.x exists, then obj.x = 50, else x = 50
    return y } } // similarly, return either obj.y, or window.y
```

We can desugar `with` by turning the comments above into code:

```
function(x, obj) {
  if (obj.hasOwnProperty("x")) { obj.x = 50 }
  else { x = 50 }
  if ("y" in obj) { return obj.y }
  else { return window.y } }
```

Soundness

Property 1 (Progress) *If σe is a closed, well-formed configuration, then either:*

- $e \in v$,
- $e = \textsf{err}\ v$, *for some* v, *or*
- $\sigma e \to \sigma' e'$, *where* $\sigma' e'$ *is a closed, well-formed configuration.*

Claim 2 (Desugar Commutes with Eval) *For all JavaScript programs* e, $desugar[\![eval_{JavaScript}(e)]\!] = eval_{\lambda_{JS}}(desugar[\![e]\!])$.

Adequacy

$$
\begin{tikzcd}
\text{JavaScript Program} \arrow[r, "desugar"] \arrow[d, "\text{Real Implementations}"'] & \lambda_{JS}\ \text{Program} \arrow[d, "\lambda_{JS}\ \text{interpreter}"] \\
\texttt{stdout} \arrow[r, leftrightarrow, "\texttt{diff}"'] & \texttt{stdout}
\end{tikzcd}
$$