PROGRAMMING LANGUAGE
SEMANTICS AS NATURAL SCIENCE

THE PECULIAR, EVOLVING, AND
BARELY CONSUMMATED RELATIONSHIP BETWEEN
SEMANTICS AND SCRIPTING LANGUAGES

Arjun Guha
Joe Gibbs Politz
Ben Lerner
Justin Pombrio
Shriram Krishnamurthi
"JavaScript has much in common with Scheme [...] Because of this deep similarity ..."
function bar(x) {
    return function() {
        var x = x;
        return x;
    };
}

var f = bar(200);
f()  \rightarrow  200

function bar(x) {
    return function() {
        var x = x;
        return x;
    };
}

var f = bar(200);
f()  \rightarrow  \text{undefined}
```javascript
var x = 0;
var y = 900;

function baz(obj) {
  with (obj) {
    x = y;
  }
}

baz({ y: 100 });
x \rightarrow 100

var myObj = { x : 0 };
baz(myObj);
x \rightarrow 100
myObj.x \rightarrow 900
```
"JavaScript has much in common with Scheme [...] Because of this deep similarity ..."

No help to researchers studying Web security, building JavaScript analyses, etc.
The Essence of JavaScript

Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi

Brown University

c = num | str | bool | undefined | null
v = c | func(x ... ) { return e } | { str: v ... }
e = x | v | let (x = e) e | e(e ... ) | e[e] | e[e] = e | de
E = ● let (x = E) e | E(e ... ) | v(v ... E, e ... )
| {str: v ... str: E, str:e ... } | E[e] | v[E] | E[e].
| v[v] = E | delete E[e] | delete v[E]

\[
\lambda_{JS} (\text{sort of}) \quad \text{on one slide}
\]

Fig. 1. Functions and Objects

\[
\begin{align*}
&\frac{\text{let (x = v) e} \leftrightarrow e[x/v] \cdots}{(\text{func}(x_1 \cdots x_n) \{ \text{return e} \})(v_1 \cdots v_n) \leftrightarrow e[x_1/1]}\\
&\{ \cdots \text{str: v } \cdots \} [\text{str}] \mapsto v
\end{align*}
\]

\[
\begin{align*}
&\frac{\text{str}_x \notin (\text{str}_1 \cdots \text{str}_n)}{\{ \text{str}_1: v_1 \cdots \text{str}_n: v_n \} [\text{str}_x] \mapsto \text{undefined}} \quad (E-\text{C})
\end{align*}
\]

\[
\begin{align*}
&\frac{\text{str}_x \notin (\text{str}_1 \cdots \text{str}_n)}{\{ \text{str}_1: v_1 \cdots \text{str}_n: v_n \} [\text{str}_x] = v_x \leftrightarrow \{ \text{str}_x: v_x, \text{str}_1: v_1 \}}
\end{align*}
\]

\[
\begin{align*}
&\text{delete} \{ \text{str}_1: v_1 \cdots \text{str}_1: v_x \cdots \text{str}_x: v_n \} [\text{str}_x] \mapsto \{ \text{str}_1: v_1 \cdots \text{str}_1: v_x \cdots \text{str}_x: v_n \}
\end{align*}
\]

\[
\begin{align*}
&\frac{\text{str}_x \notin (\text{str}_1 \cdots \text{str}_n)}{\text{delete} \{ \text{str}_1: v_1 \cdots \} [\text{str}_x] \mapsto \{ \text{str}_1: v_1 \cdots \}} \quad (E-\text{DE})
\end{align*}
\]

Fig. 4. Prototype-Based Objects

We use \( \rightarrow \) to denote the reflexive transitive closure of \( \rightarrow \):

\[
\begin{align*}
&\frac{\text{str}_x \notin (\text{str}_1 \cdots \text{str}_n)}{\{ \text{str}_1: v_1, \cdots, \text{str}_n: v_n \} [\text{str}_x] \mapsto \text{undefined}} \quad (E-\text{FIELD-NOTFOUND})
\end{align*}
\]

\[
\begin{align*}
&\frac{\text{str}_x \notin (\text{str}_1 \cdots \text{str}_n)}{\{ \text{str}_1: v_1 \cdots \text{"_\text{proto}_--"}: \text{null } \cdots \text{str}_n: v_n \} [\text{str}_x] \mapsto \text{undefined}} \quad (E-\text{FIELD-PROTO-NULL})
\end{align*}
\]

\[
\begin{align*}
&\frac{\text{str}_x \notin (\text{str}_1 \cdots \text{str}_n)}{\text{str}_x \notin (\text{str}_1 \cdots \text{str}_n)} \quad p = \text{ref } l
\end{align*}
\]

\[
\begin{align*}
&\frac{\{ \text{str}_1: v_1 \cdots \text{"_\text{proto}_--"}: p \cdots \text{str}_n: v_n \} [\text{str}_x] \mapsto (\text{deref } p) [\text{str}_x]} \quad (E-\text{FIELD-PROTO})
\end{align*}
\]
JavaScript program \( \xrightarrow{\text{desugar}} \) \( \lambda_{\text{JS}} \) program

SpiderMonkey, V8, Rhino \( \xrightarrow{\text{definitional interpreter}} \) "our answer"

Identical for conformance suites

"their answer"
What About the Spec?

1. The spec is embodied in the implementations
2. The spec is incomplete: e.g., SES depends on `window.console`
3. The spec depends on implementations!
   
   *If [...], the behavior of `sort` is implementation-defined.*

4. Attackers attack implementations, not specs
Internal/External validation
TWO POSITIONS

1. The desugar/semantics split is vital

2. Tests are a form of specification
1. Curated “essence” — provides insight
2. Target for proofs
3. Target for tools
4. Stabilizes quickly and rarely changes after
5. What we as scientists should do
Tests as Specifications

Tests are incomplete but formal
Implementations on their own over-specify
Tests keep up with evolution
Tests ease the interface with specification authors
THREE RESEARCH PROBLEMS
1. Shrinking Desugaring Output
let (%context = %nonstrictContext) {
  %defineGlobalAccessors(%context, "n");
  %defineGlobalAccessors(%context, "x");
  let (#strict = false) {
    try {
      %set-property(%ToObject(%context["x", {
        #proto: null,
        #class: "Object",
        #extensible: true,
      }]), "count",
      %PrimAdd(%context["n", {
        #proto: null,
        #class: "Object",
        #extensible: true,
      }], 1.))
    } catch {
      %ErrorDispatch
    }
  }
}

x["count"] = n + 1;
let (\%context = \%nonstrictContext) {
  \%defineGlobalAccessors(\%context, "n");
  \%defineGlobalAccessors(\%context, "x");
  let (#\text{strict} = \text{false}) {
    try {
      \%set-property(
        \%ToObject(
          \%context["x", {
            \#proto: null,
            \#class: "Object",
            \#extensible: true,\}]),
          "count",
          \%PrimAdd(\%context["n", {
            \#proto: null,
            \#class: "Object",
            \#extensible: true,\}]],
          1.))
    } catch {
      \%ErrorDispatch
    }
  }
}
try {
  %set-property(
    %ToObject(
      %context["x", {{#proto: null,
                     #class: "Object",
                     #extensible: true,}}]),
      "count",
      %PrimAdd(%context["n" , {{#proto: null,
                                #class: "Object",
                                #extensible: true,}}],
               1.))
  )
} catch {
  %ErrorDispatch
}
x["count"] = n + 1;

1. Dead-code elimination
2. Constant propagation
3. Type-driven specialization
2. LIFTING DESUGARING THROUGH REDUCTIONS
Three key properties:

1. Emulation
   Desugaring a re-sugared term yields the same desugared term

2. Abstraction
   Re-sugaring does not show terms introduced by desugaring

3. Completeness
   Doesn’t skip expected steps
3. Reducing Effort Per Semantics
<table>
<thead>
<tr>
<th>Artifact</th>
<th>Effort</th>
<th>People</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cisco IOS</td>
<td>1y x 2</td>
<td>1 PhD, 1 MS</td>
</tr>
<tr>
<td>EcmaScript 3</td>
<td>3m x 2</td>
<td>1 PhD, 1 UG</td>
</tr>
<tr>
<td>EcmaScript 5 Safe</td>
<td>5m x 4</td>
<td>1 PD, 2 PhD, 1 MS</td>
</tr>
<tr>
<td>DOM Events</td>
<td>7m x 4</td>
<td>1 PD, 1 PhD, 1 MS, 1 UG</td>
</tr>
</tbody>
</table>

Python: The Full Monty
A Tested Semantics for the Python Programming Language

Joe Gibbs Politz  
Providence, RI, USA  
joe@cs.brown.edu

Alejandro Martinez  
La Plata, BA, Argentina  
amtriathlon@gmail.com

Matthew Milano  
Providence, RI, USA  
matthew@cs.brown.edu

Sumner Warren  
Providence, RI, USA  
jswarren@cs.brown.edu

Daniel Patterson  
Providence, RI, USA  
dbpatter@cs.brown.edu

Junsong Li  
Beijing, China  
ljs.darkfish@gmail.com

Anand Chitipothu  
Bangalore, India  
anandology@gmail.com

Shriram Krishnamurthi  
Providence, RI, USA  
sk@cs.brown.edu
New languages with JVM implementations

- **Alef++**, a language inspired by Perl and Lisp.[17]
- **Ateji PX**, an extension of Java for easy parallel programming on multicore, GPU, Grid and Cloud.[18]
- **BBj**, an object-oriented language for business applications
- **BeanShell**, a scripting language whose syntax is close to Java.
- **Ceylon**, an upcoming Red Hat's Java competitor
- **ColdFusion**, a scripting language compiled to Java, used on the ColdFusion application Server
- **CAL**, a Haskell-inspired functional language.
- **E** language has an implementation on the JVM.
- **Fantom**, a language built from the base to be portable across the JVM, .NET CLR, and JavaScript.[19]
- **Flow Java**.
- **Fortress**, a language designed by Sun as a successor to Fortran, mainly for parallel scientific computing.
- **Fregé**, a non-strict, pure functional programming language in the spirit of Haskell.[20]
- **Frink**, a language that tracks units of measure through calculations.
- **Gosu**, an extensible type-system language compiled to Java bytecode.
- **Hecl,[21]**
- **Io**ke, a prototype-based language somewhat reminiscent of Io, with similarities to Ruby, Lisp and Smalltalk.
- **KBML**, an expert system DSL for defining correlation rules and event processing. Used by products based on the OpenKBM platform.
- **Kotlin (programming language)** invented by JetBrains
- **Jabaco**, A BASIC-like GUI RAD language for Windows that uses the JVM.
- **Jaskell**, a Haskell inspired scripting language.[22]
- **Jelly**.
- **Join Java**, a language that extends Java with the join semantics of the join-calculus.
- **Joy**.
- **JudsonScript**.
- **Libretto**, Dynamic general purpose object-oriented programming language.[23]
- **Mirah**, a customizable language featuring type inference and a highly Ruby-inspired syntax.[24]
- **N.A.M.E. Basic**.
- **NetLogo**, a multi-agent language.
- **Nice**.
- **Noop**, a language built with testability as a major focus.
- **ObjectScript**.
- **PHP.reboot**, a PHP-style language.[25]
- **Pizza**, a superset of Java with function pointers and algebraic data types.
- **Pnuts**.
- **Stab**, a C# work-alike.[26]
- **Sleep**, a procedural scripting language inspired by Perl and Objective-C.
- **V** language has an implementation on the JVM.[27]
- **Xtend**, a language built by the Eclipse foundation, featuring very tight Java interoperability, with a focus on extension methods and lambdas, and rich tooling
- **X10**, a language designed by IBM, featuring constrained types and a focus on concurrency and distribution.
- **Yeti**, a ML style functional language, that runs on the JVM.[28]
Not Just “Languages”

Environments, APIs, event models define behavior

Where do we get the next 700 semantics?
Assume:

- **P parser** (*ripped out*)
- Candidate $\lambda_p$
- $\lambda_p$ evaluator

"their answer" \[\quad\quad\quad\textit{want it to be identical}\quad\quad\quad\text{"our answer"} \]
Learn this using machine translation techniques.
Important Differences

MT Tree Alignment needs:
• Lots of sentences of input language (easy)
• Lots of sentences of output language (easy)
• Lots of examples of translations (oops!)
  Typically at least 1mil, preferably 10mil

But MT also lacks something we have…
These represent ground truth.
Current Status

We’ve tried four different approaches:

• Naïve tree matching
• Tree transducer by Gibbs sampling
• Genetic programming
• Sketching

None has yet succeeded beyond toy examples
Summary

• The purpose of a semantics is insight, not only matching execution behavior

• Decomposing into desugaring and a core semantics offers room for flexibility

• Desugaring deserves more respect in semantics research

• Tests are underutilized in semantics
The Modelers’ Hippocratic Oath
Emmanuel Derman and Paul Wilmott

I will remember that I didn't make the world, and it doesn't satisfy my equations.

Though I will use models boldly to estimate value, I will not be overly impressed by mathematics.

I will never sacrifice reality for elegance without explaining why I have done so.

Nor will I give the people who use my model false comfort about its accuracy. Instead, I will make explicit its assumptions and oversights.

I understand that my work may have enormous effects on society and the economy, many of them beyond my comprehension.