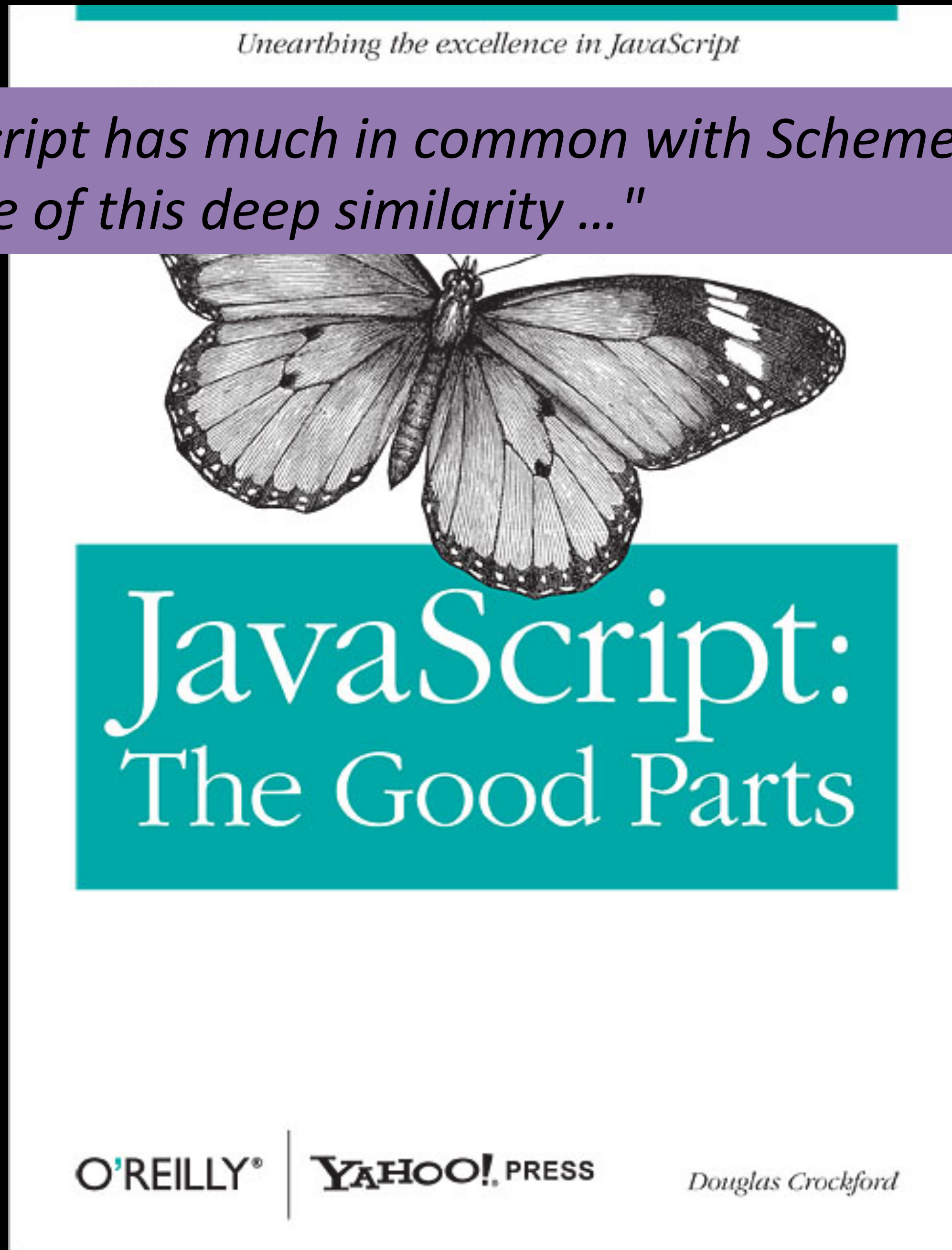
Programming Language Semantics as Natural Science

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 ..."

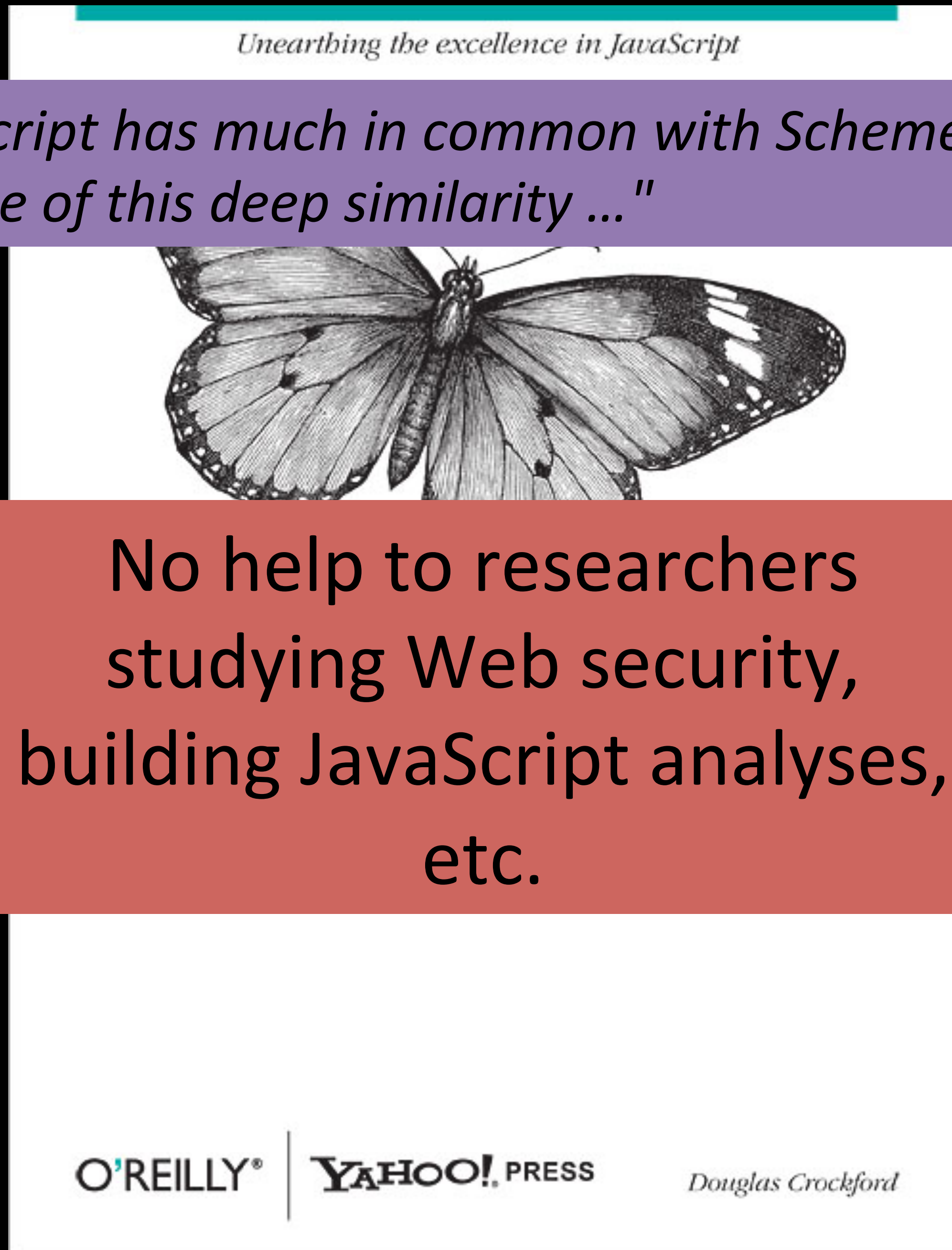


```
function bar(x) {
  return function() {
    var x = x;
    return x;
  };
}

var f = bar(200);
f()
→ 200
```

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

var f = bar(200);
f()
→ undefined
```

```
var x = 0;
var y = 900;

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

baz({ y: 100 });
```
x → 100

```
var myObj = { x : 0 };
baz(myObj);
```
x → 100
myObj.x → 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

λ_{JS} (sort of) on one slide

$$\begin{aligned}
c &= num \mid str \mid bool \mid \textbf{undefined} \mid \textbf{null}\\
v &= c \mid \textbf{func}(x\cdots)\ \{\ \textbf{return}\ e\ \} \mid \{\ str{:}v\cdots\ \}\\
e &= x \mid v \mid \textbf{let}\ (x = e)\ e \mid e(e\cdots) \mid e[e] \mid e[e] = e \mid \textbf{de}\\
E &= \bullet \mid \textbf{let}\ (x = E)\ e \mid E(e\cdots) \mid v(v\cdots\ E,\ e\cdots)\\
&\mid \{str{:}\ v\cdots\ str{:}E,\ str{:}e\cdots\ \} \mid E[e] \mid v[E] \mid E[e]\\
&\mid v[v] = E \mid \textbf{delete}\ E[e] \mid \textbf{delete}\ v[E]
\end{aligned}$$

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

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

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

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

$$\{\ str_1{:}\ v_1\cdots\ str_i{:}\ v_i\ \cdots str_n{:}\ v_n\ \}\ [str_i]$$
$$\hookrightarrow \{\ str_1{:}\ v_1\cdots\ str_i{:}\ v\ \cdots str_n{:}\ v_n\ \}$$

$$\frac{str_x \notin (str_1\cdots)}{\{\ str_1{:}\ v_1\cdots\ \}\ [str_x] = v_x \hookrightarrow \{\ str_x{:}\ v_x,\ str_1{:}\ v_1}$$

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

$$\frac{str_x \notin (str_1\cdots)}{\textbf{delete}\ \{\ str_1{:}\ v_1\cdots\ \}\ [str_x] \hookrightarrow \{\ str_1{:}\ v_1\cdots\ \}}\ \text{(E-De}$$

Fig. 1. Functions and Objects

$l = \cdots$	Locations
$v = \cdots \mid l$	Values
$\sigma = (l, v)\cdots$	Stores
$e = \cdots \mid e = e \mid \textbf{ref}\ e \mid \textbf{deref}\ e$	Expressions
$E = \cdots \mid E = e \mid v = E \mid \textbf{ref}\ E \mid \textbf{deref}\ E$	Evaluation Contexts

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

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

$$\sigma E\langle \textbf{deref}\ l\rangle \to \sigma E\langle \sigma(l)\rangle \quad \text{(E-Deref)}$$

$$\sigma E\langle l = v\rangle \to \sigma[l/v]E\langle l\rangle \quad \text{(E-SetRef)}$$

We use $\twoheadrightarrow$ to denote the reflexive-transitive closure of $\to$.

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

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

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

Fig. 4. Prototype-Based Objects

JavaScript program

desugar

λ_{JS} program

SpiderMonkey, V8, Rhino

definitional interpreter

Identical for conformance suites

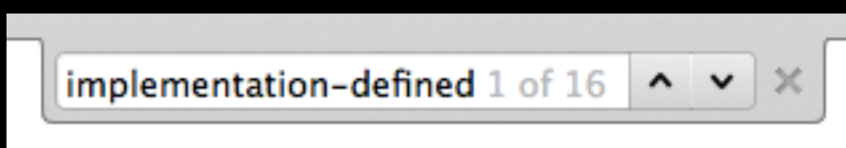
"their answer"

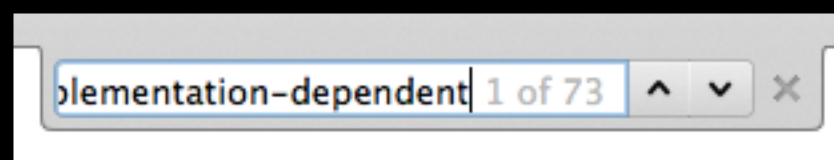
"our answer"

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!

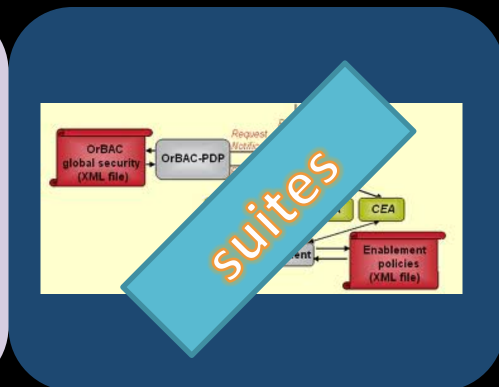
   If [...], the behavior of **`sort`**
   is implementation-defined.





4. Attackers attack implementations, not specs

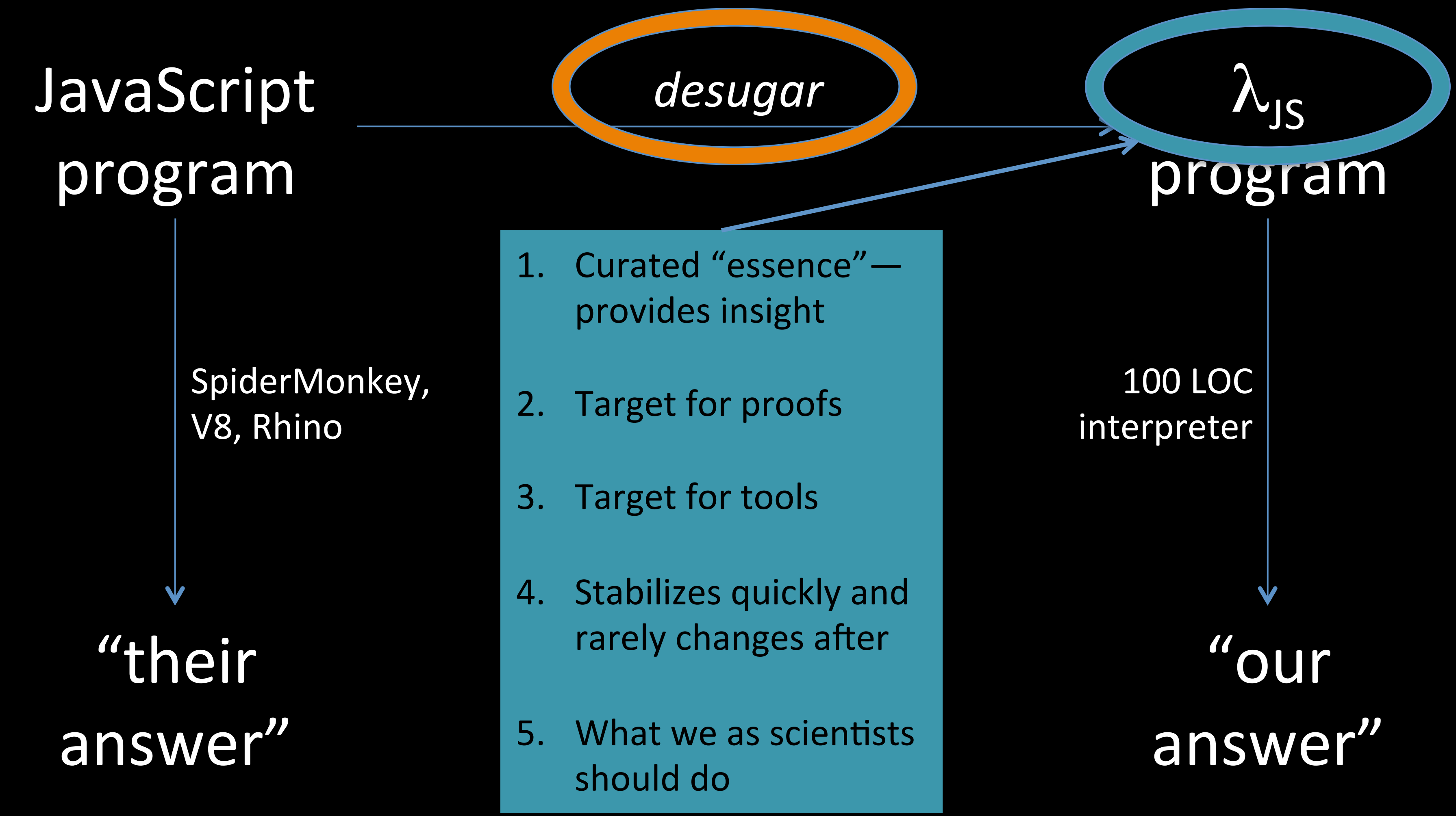
Internal/
External
validation



Two Positions

1. The desugar/semantics split is vital

2. Tests are a form of specification


JavaScript program
desugar
λJS program
SpiderMonkey, V8, Rhino
100 LOC interpreter
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
"their answer"
"our answer"

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

```
x["count"] = n + 1;
```

```
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
    }
  }
}
```

```
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
    }
  }
}
```

1. Strict mode
2. No access to globals

```
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
}
```

No inspection of missing properties

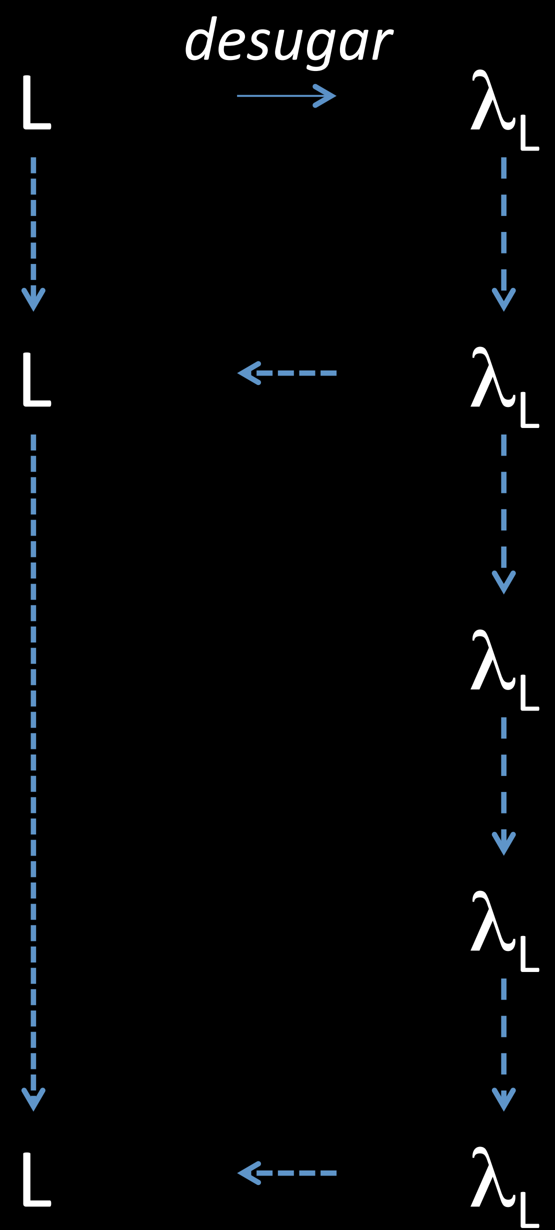
```
%set-property(
  %ToObject(
    %context["x", {[#proto: null,
                    #class: "Object",
                    #extensible: true,]}]),
  "count",
  %PrimAdd(%context["n" , {[#proto: null,
                            #class: "Object",
                            #extensible: true,]}],
           1.))
```

```
x["count"] = n + 1;
```

```
%set-property(
  %ToObject(
    %context["x", {[#proto: null,
                    #class: "Object",
                    #extensible: true,]}]),
  "count",
  %PrimAdd(%context["n" , {[#proto: null,
                            #class: "Object",
                            #extensible: true,]}],
         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

Artifact	Effort	People
Cisco IOS	1y x 2	1 PhD, 1 MS
EcmaScript 3	3m × 2	1 PhD, 1 UG
EcmaScript 5 Safe	5m × 4	1 PD, 2 PhD, 1 MS
DOM Events	7m × 4	1 PD, 1 PhD, 1 MS, 1 UG

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

Ruby

DART

the programming language
Lua

Groovy

python

ERLANG

HipHop
for
php

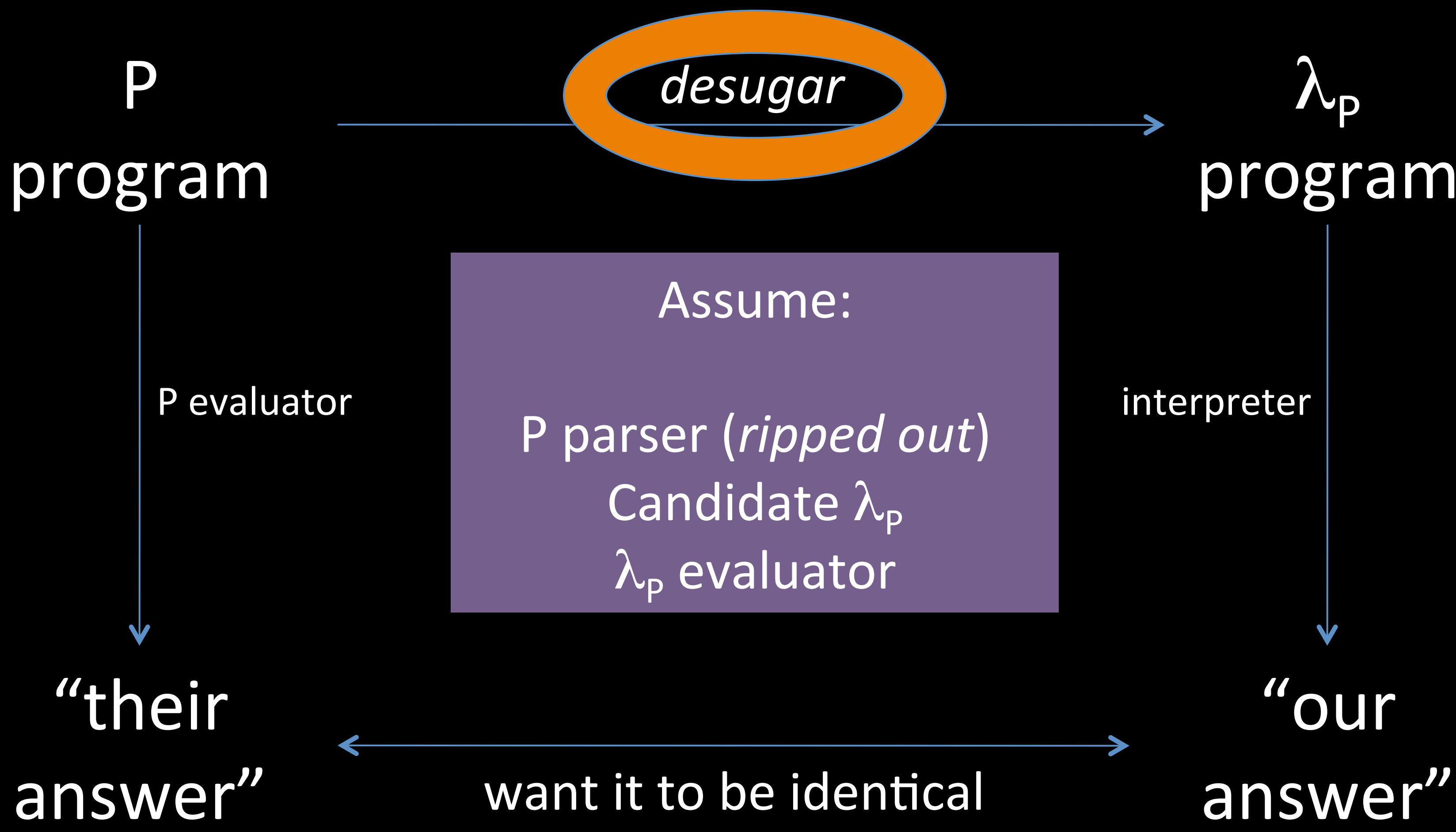
New languages with JVM implementations

- 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.
- Frege, 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]
- Ioke, 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.
- Judoscript.
- 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]

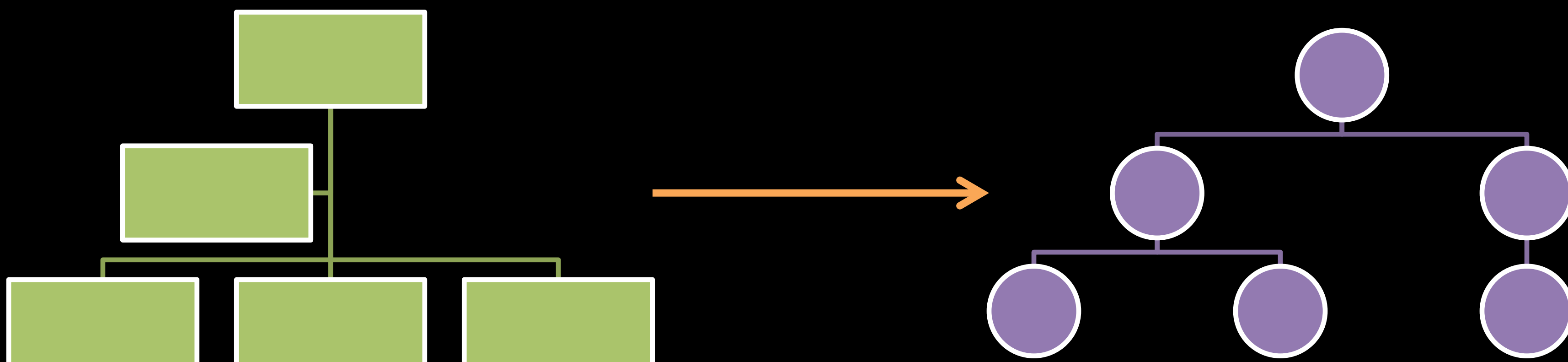
Not Just "Languages"

Environments, APIs, event models define behavior

Where do we get the next 700 semantics?


P
program
desugar
λP
program
P evaluator
Assume:
P parser (ripped out)
Candidate λP
λP evaluator
interpreter
“their
answer”
want it to be identical
“our
answer”

P program $\xrightarrow{\textit{desugar}}$ λ_P program



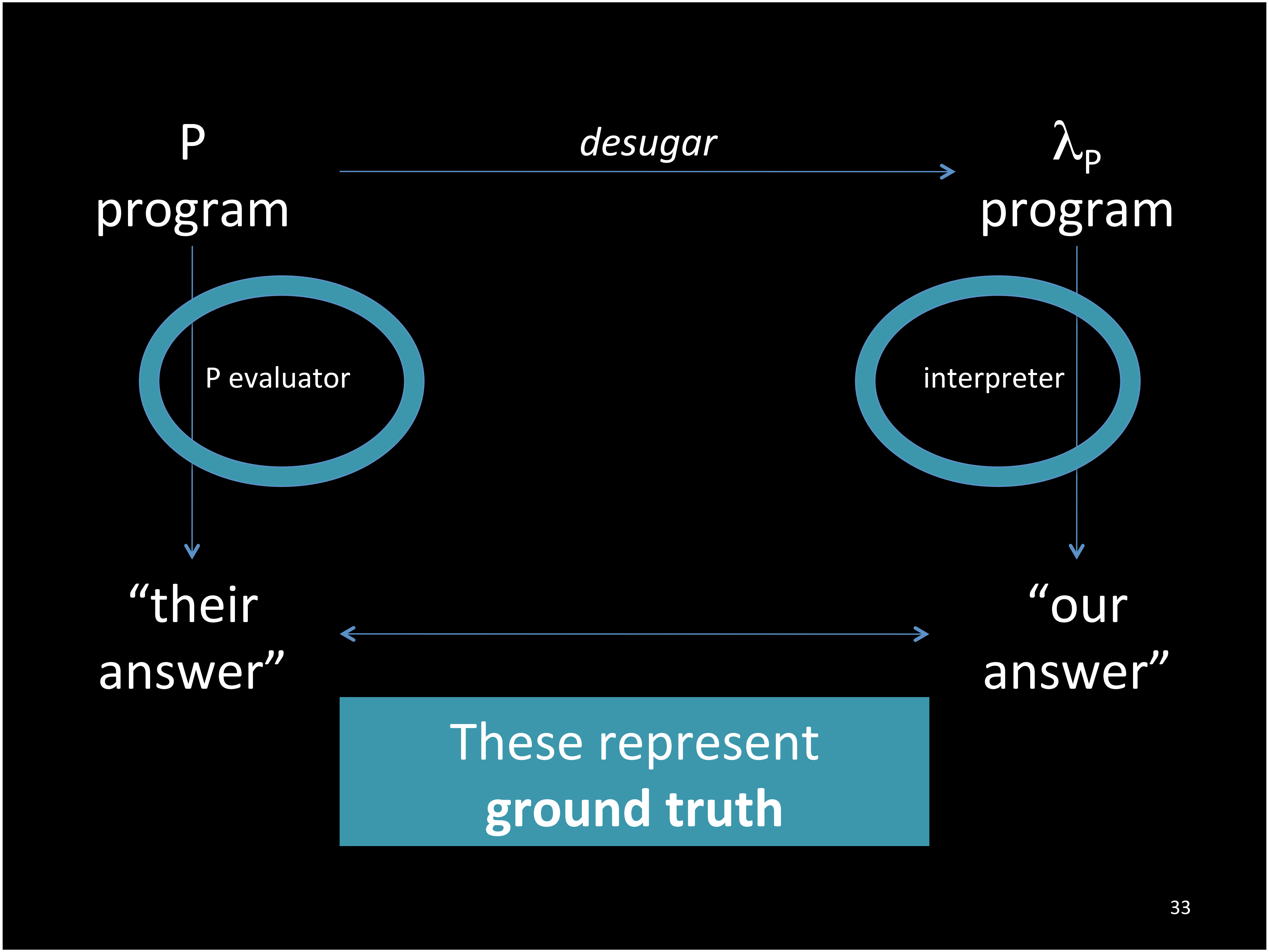
Learn this using
machine translation
techniques

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…


P
program
desugar
λ_P
program
P evaluator
interpreter
"their
answer"
"our
answer"
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.

http://www.wilmott.com/blogs/eman/index.cfm/2009/1/8/The-Financial-Modelers-Manifesto