KJS: A Complete Formal Semantics of JavaScript

Daejun Park Andrei Stefanescu Grigore Rosu
University of Illinois at Urbana-Champaign
June 16, 2015 @ PLDI'15

Why semantics matters?

```
var _ = function f() {

          f();
        };

f();
```

```
function f() {

  f();
}

f();
```

Why semantics matters?

anonymous function expression

```
var _ = function f() {

          f();
        };

f();
```

function declaration

```
function f() {

  f();
}

f();
```

Why semantics matters?

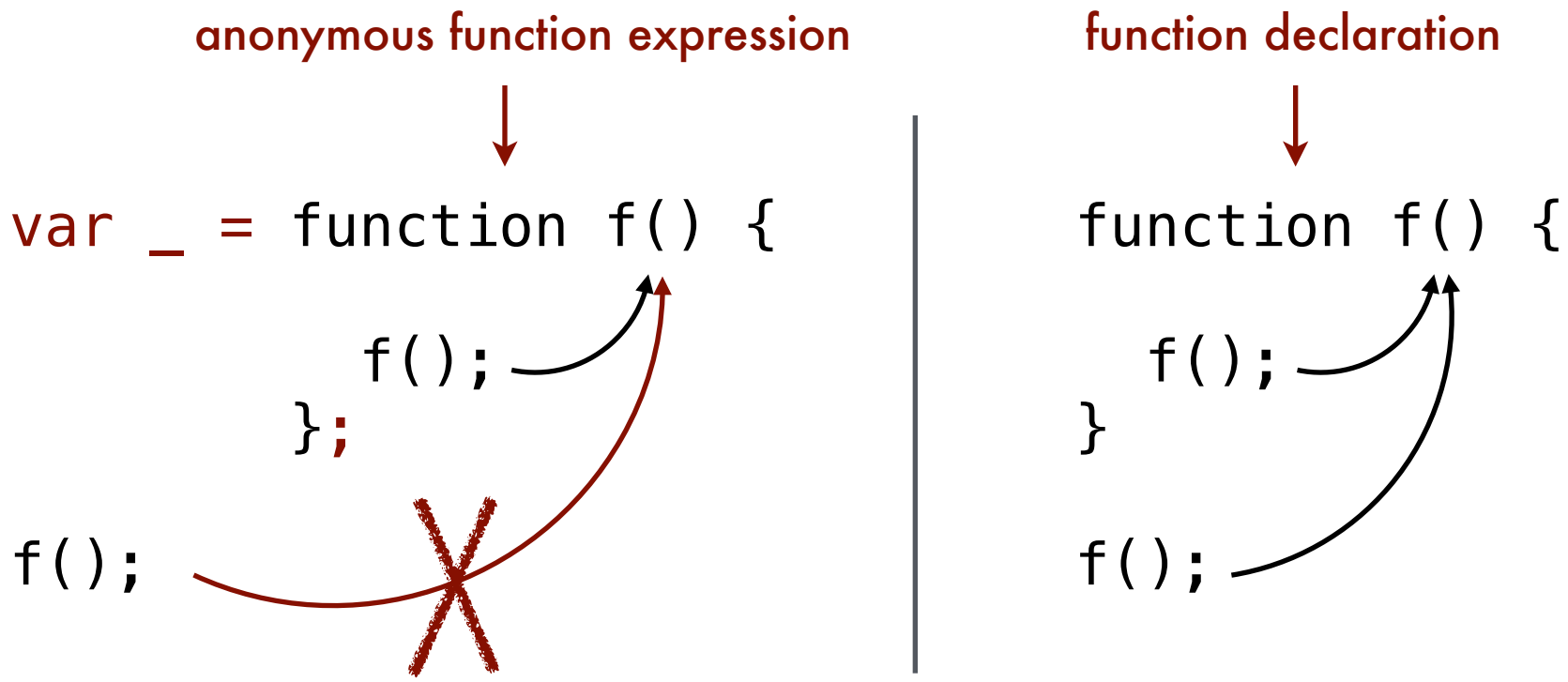
anonymous function expression

```
var _ = function f() {
          f();
        };

f();
```

function declaration

```
function f() {
  f();
}

f();
```

Why semantics matters?



Why semantics matters?

```
"use strict";
var _ = function f() {
          f = 0;
        };
```

← runtime error

```
"use strict";
function f() {
  f = 0;

}
```

← no error

Why semantics matters?

Chrome 38.0 and Safari 7.0.4 failed to conform to standard.
Fixed in Chrome 41.0 and Safari 8.0.6

```
"use strict";
var _ = function f() {
          f = 0; ← runtime error in Firefox, but
                   silently ignored in Chrome and Safari.
        };
```

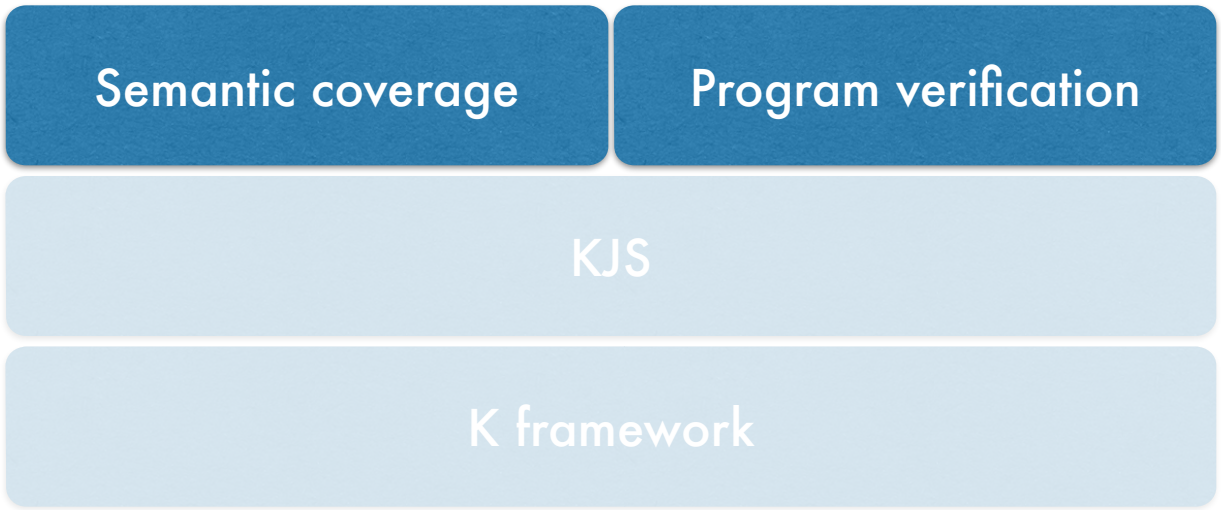
Overview

Overview

K framework
(kframework.org)

Overview

Overview



K framework [Rosu and Serbanuta 2010]

Language semantics engineering framework (kframework.org)

Syntax. BNF annotated with evaluation strategy.

Semantics. (modular) small step operational semantics.
i.e., a set of reduction rules over program states

$$S \Rightarrow S'$$

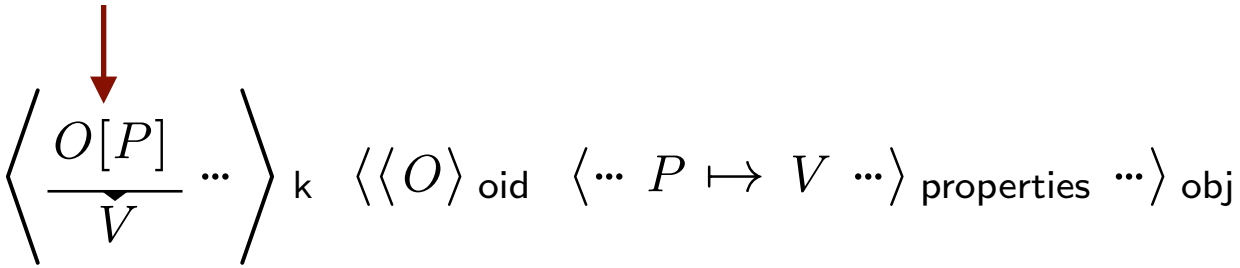
Semantic rules in K

Rule for object field lookup:

$$\left\langle \frac{O[P]}{V} \cdots \right\rangle_{\mathsf{k}} \quad \langle \langle O \rangle_{\mathsf{oid}} \quad \langle \cdots \; P \mapsto V \; \cdots \rangle_{\mathsf{properties}} \; \cdots \rangle_{\mathsf{obj}}$$

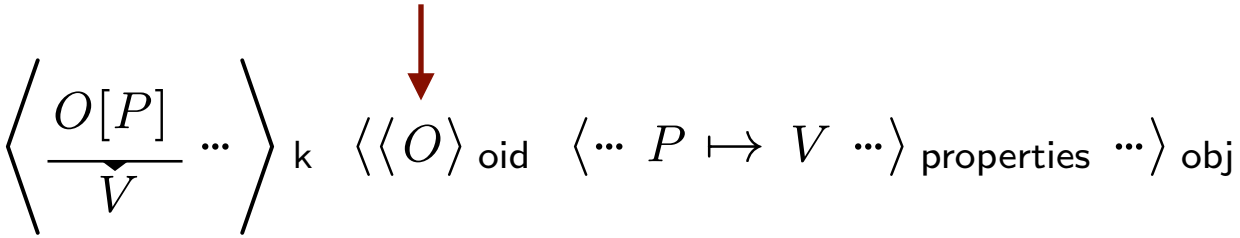
Semantic rules in K

Rule for object field lookup:



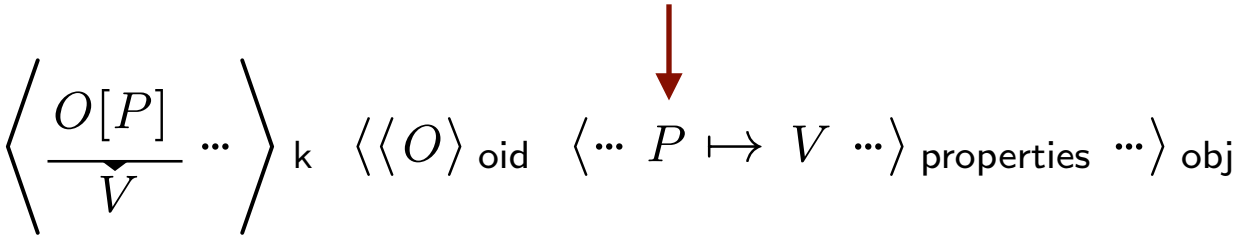
Semantic rules in K

Rule for object field lookup:



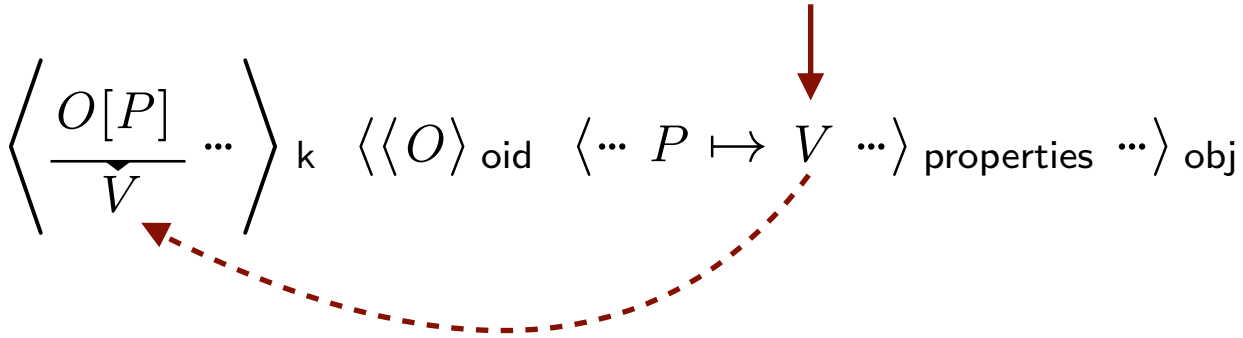
Semantic rules in K

Rule for object field lookup:



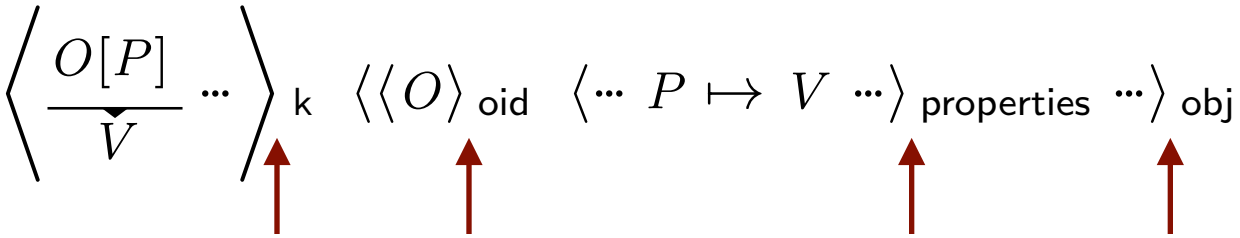
Semantic rules in K

Rule for object field lookup:



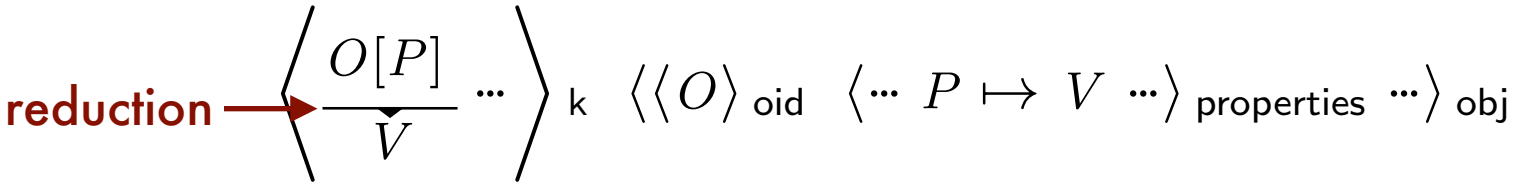
Semantic rules in K

Rule for object field lookup:



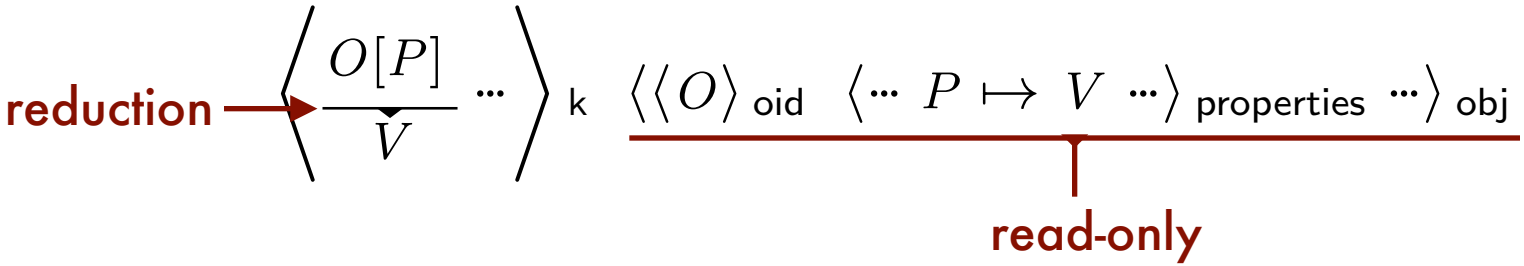
Semantic rules in K

Rule for object field lookup:



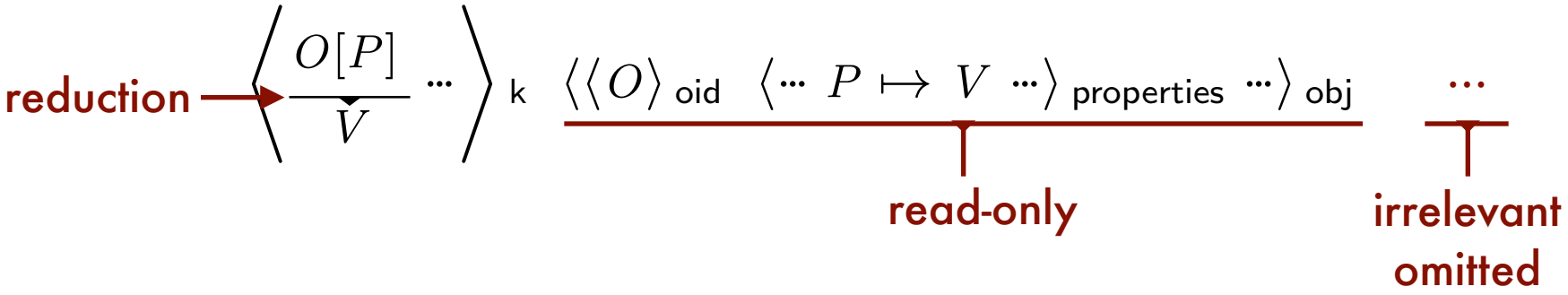
Semantic rules in K

Rule for object field lookup:

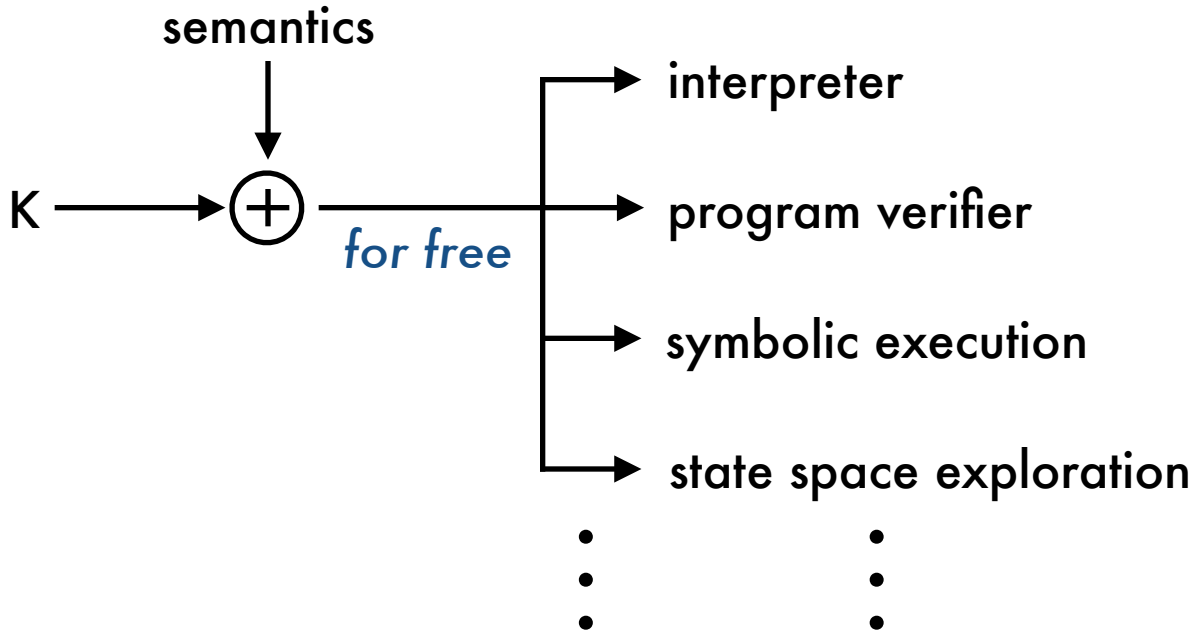


Semantic rules in K

Rule for object field lookup:



Semantic-driven formal analysis



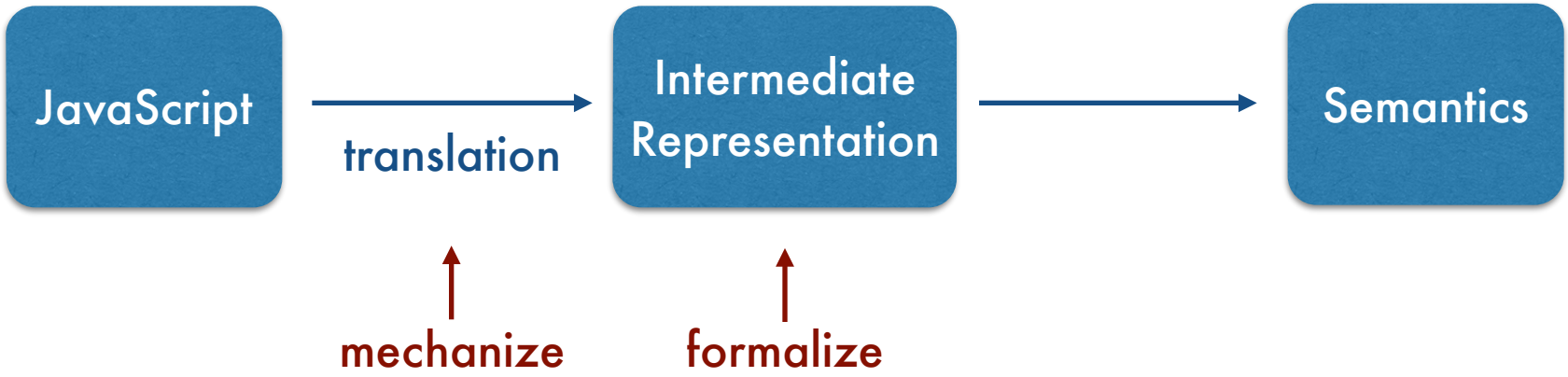

KJS
K framework

KJS: outline

KJS *faithfully* formalizes ECMAScript 5.1 (informal) standard.

KJS: outline

KJS *faithfully* formalizes ECMAScript 5.1 (informal) standard.



KJS: one-to-one mapping to standard

The expression "++ *Expression*" is evaluated as follows:

1. Let *expr* be the result of evaluating *Expression*.
2. Let *oldValue* be ToNumber(GetValue(*expr*)).
3. Let *newValue* be the result of adding the value `1` to *oldValue*.
4. Call PutValue(*expr*, *newValue*).
5. Return *newValue*.

ECMAScript 5.1 standard

KJS: one-to-one mapping to standard

The expression "++ *Expression*" is evaluated as follows:

1. Let *expr* be the result of evaluating *Expression*.
2. Let *oldValue* be ToNumber(GetValue(*expr*)).
3. Let *newValue* be the result of adding the value `1` to *oldValue*.
4. Call PutValue(*expr*, *newValue*).
5. Return *newValue*.

ECMAScript 5.1 standard

```
rule ++ Expression =>
 Let $expr = @GetReference(Expression);
 Let $oldValue = ToNumber(GetValue($expr));
 Let $newValue = @Addition($oldValue,1);
 Do PutValue($expr,$newValue);
 Return $newValue;
```

KJS

KJS: one-to-one mapping to standard

The expression "++ *Expression*" is evaluated as follows:

1. Let *expr* be the result of evaluating *Expression*.
2. Let *oldValue* be ToNumber(GetValue(*expr*)).
3. Let *newValue* be the result of adding the value `1` to *oldValue*.
4. Call PutValue(*expr*, *newValue*).
5. Return *newValue*.

ECMAScript 5.1 standard

```
rule ++ Expression =>
 Let $expr = @GetReference(Expression);
 Let $oldValue = ToNumber(GetValue($expr));
 Let $newValue = @Addition($oldValue,1);
 Do PutValue($expr,$newValue);
 Return $newValue;
```

KJS

KJS: one-to-one mapping to standard

each step of informal description → systematic translation

The expression "++ *Expression*" is evaluated as follows:

1. Let *expr* be the result of evaluating *Expression*.
2. Let *oldValue* be ToNumber(GetValue(*expr*)).
3. Let *newValue* be the result of adding the value `1` to *oldValue*.
4. Call PutValue(*expr*, *newValue*).
5. Return *newValue*.

ECMAScript 5.1 standard

```
rule ++ Expression =>
 Let $expr = @GetReference(Expression);
 Let $oldValue = ToNumber(GetValue($expr));
 Let $newValue = @Addition($oldValue,1);
 Do PutValue($expr,$newValue);
 Return $newValue;
```

KJS

KJS: one-to-one mapping to standard

each step of informal description → formal pseudo-code statement

systematic translation

The expression "++ *Expression*" is evaluated as follows:

1. Let *expr* be the result of evaluating *Expression*.
2. Let *oldValue* be ToNumber(GetValue(*expr*)).
3. Let *newValue* be the result of adding the value `1` to *oldValue*.
4. Call PutValue(*expr*, *newValue*).
5. Return *newValue*.

ECMAScript 5.1 standard

```
rule ++ Expression =>
 Let $expr = @GetReference(Expression);
 Let $oldValue = ToNumber(GetValue($expr));
 Let $newValue = @Addition($oldValue,1);
 Do PutValue($expr,$newValue);
 Return $newValue;
```

KJS

KJS: one-to-one mapping to standard

each step of informal description ⟷ formal pseudo-code statement

1-to-1 mapping

systematic translation

The expression "++ *Expression*" is evaluated as follows:

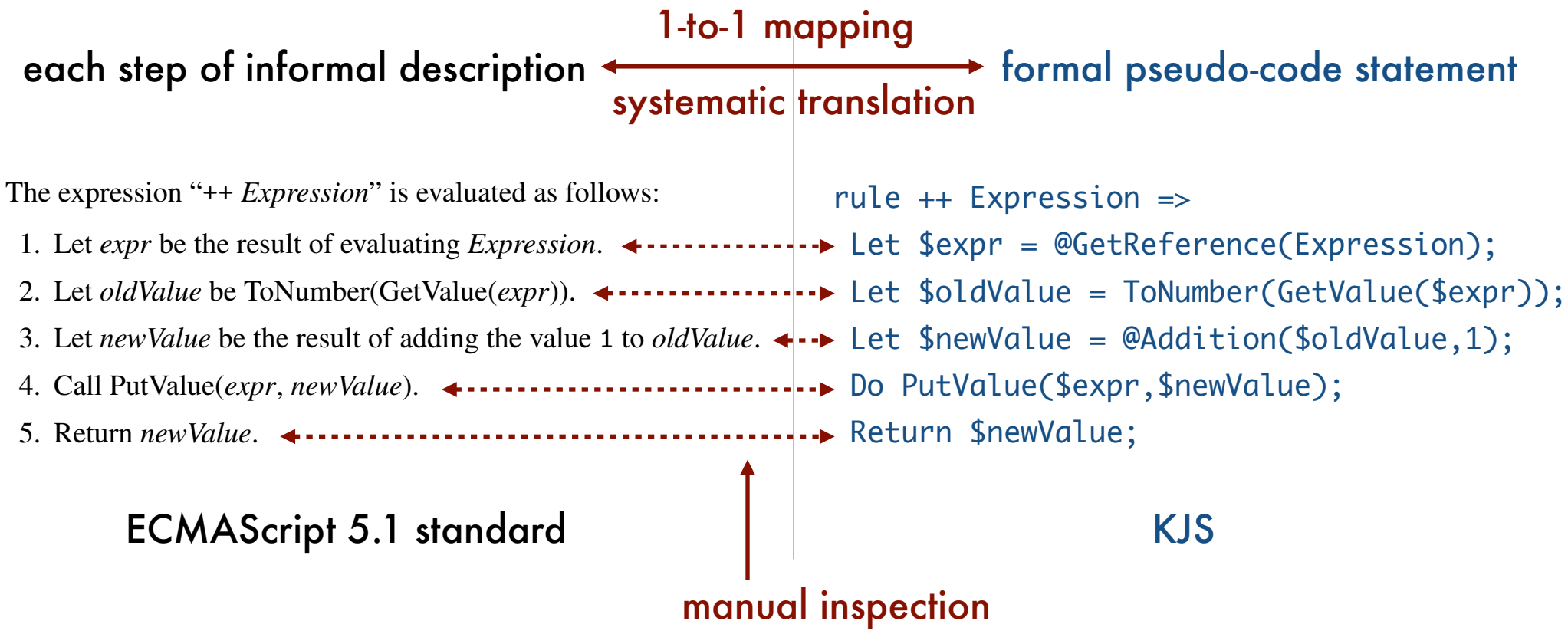
1. Let *expr* be the result of evaluating *Expression*.
2. Let *oldValue* be ToNumber(GetValue(*expr*)).
3. Let *newValue* be the result of adding the value 1 to *oldValue*.
4. Call PutValue(*expr*, *newValue*).
5. Return *newValue*.

```
rule ++ Expression =>
Let $expr = @GetReference(Expression);
Let $oldValue = ToNumber(GetValue($expr));
Let $newValue = @Addition($oldValue,1);
Do PutValue($expr,$newValue);
Return $newValue;
```

ECMAScript 5.1 standard

KJS

KJS: one-to-one mapping to standard



Completeness

Tested against ECMAScript conformance test suite.

Formal Semantics	Passed	Failed	% passed
KJS	2,782	0	100.0%
[Politz et al. 2012][5]	2,470	345	87.7%
[Bodin et al. 2014]	1,796	986	64.6%

Most complete semantics to date.

Completeness

Tested against ECMAScript conformance test suite.

Formal Semantics	Passed	Failed	% passed
KJS	2,782	0	100.0%
[Politz et al. 2012][5]	2,470	345	87.7%
[Bodin et al. 2014]	1,796	986	64.6%

Most complete semantics to date.

Completeness

Tested against ECMAScript conformance test suite.

Formal Semantics	Passed	Failed	% passed
KJS	2,782	0	100.0%
[Politz et al. 2012][5]	2,470	345	87.7%
[Bodin et al. 2014]	1,796	986	64.6%



Most complete semantics to date.

Development cost

Took only *four months* by a first year PhD student.

semantic rules: 1,370

Development cost

Took only *four months* by a first year PhD student.

semantic rules: 1,370

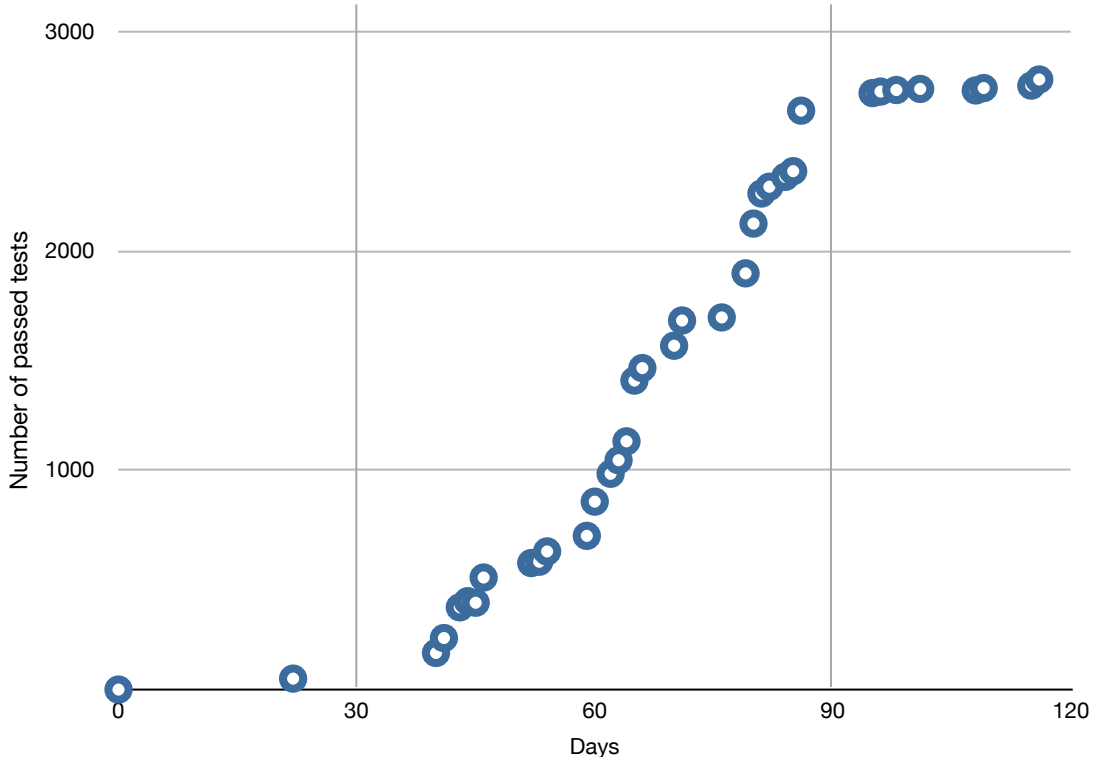
Thanks to:

- K's executability
- Systematic translation
- K's modularity

Development cost

Took only *four months* by a first year PhD student.

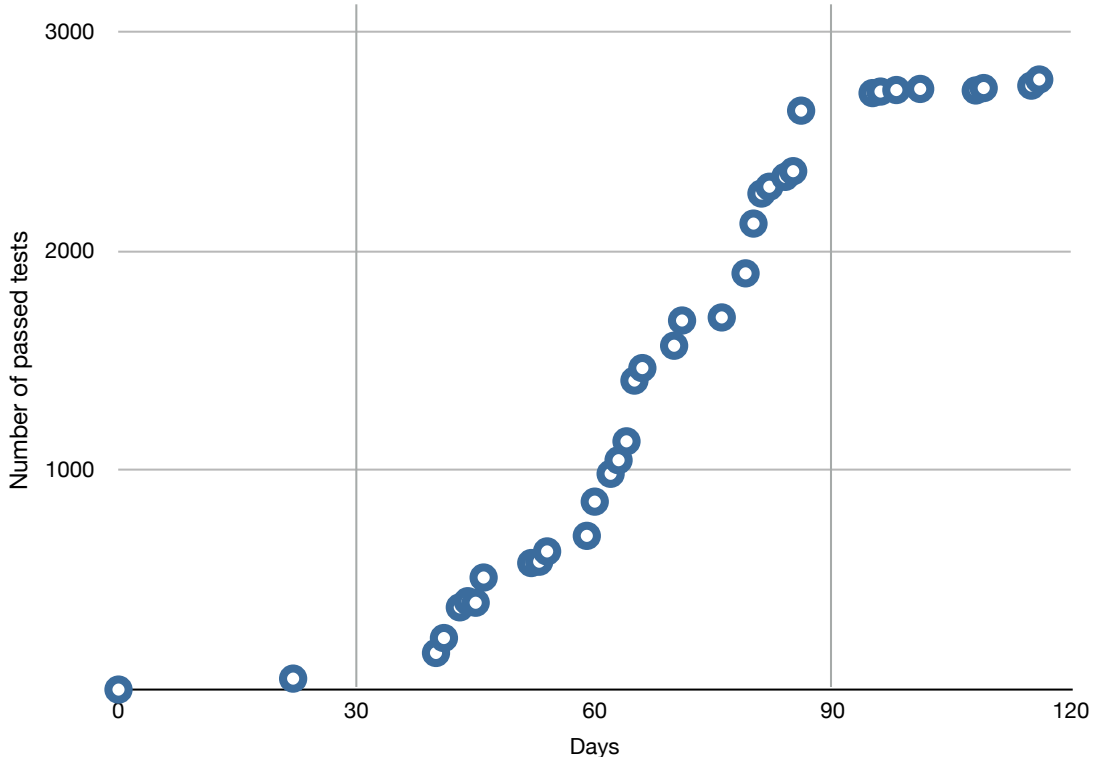
semantic rules: 1,370



Development cost

Took only *four months* by a first year PhD student.

semantic rules: 1,370

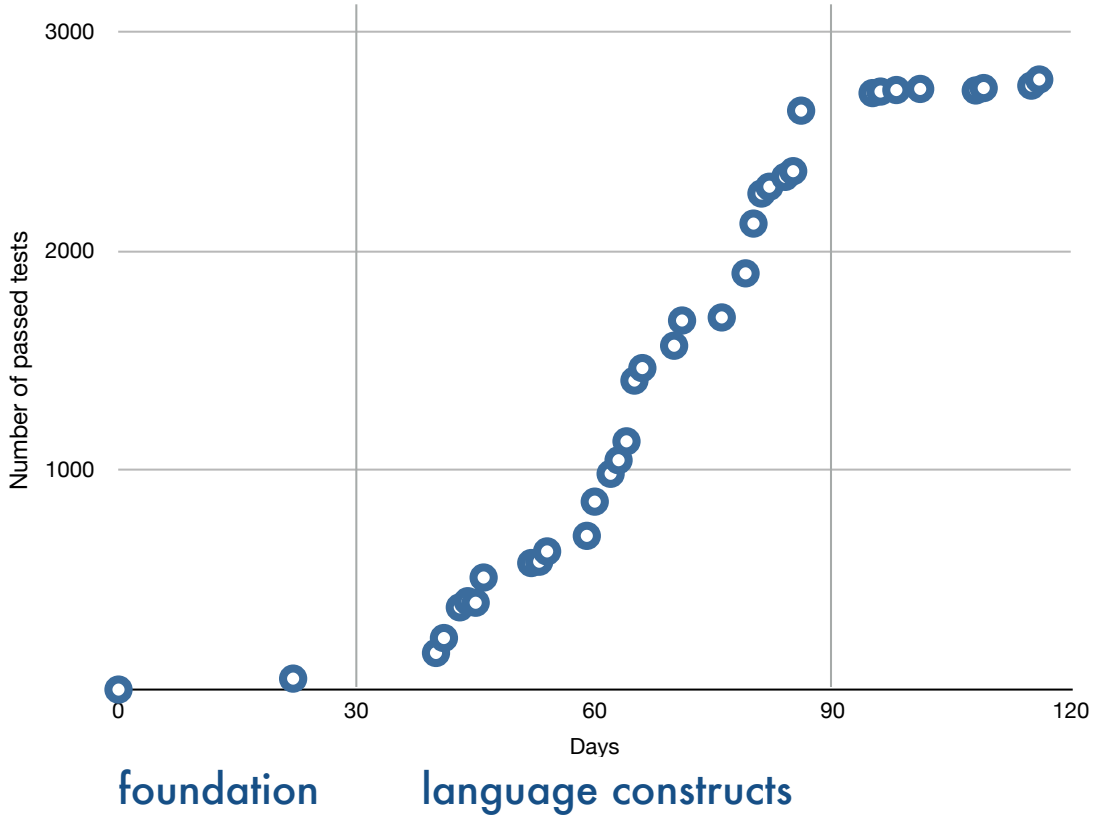


foundation

Development cost

Took only *four months* by a first year PhD student.

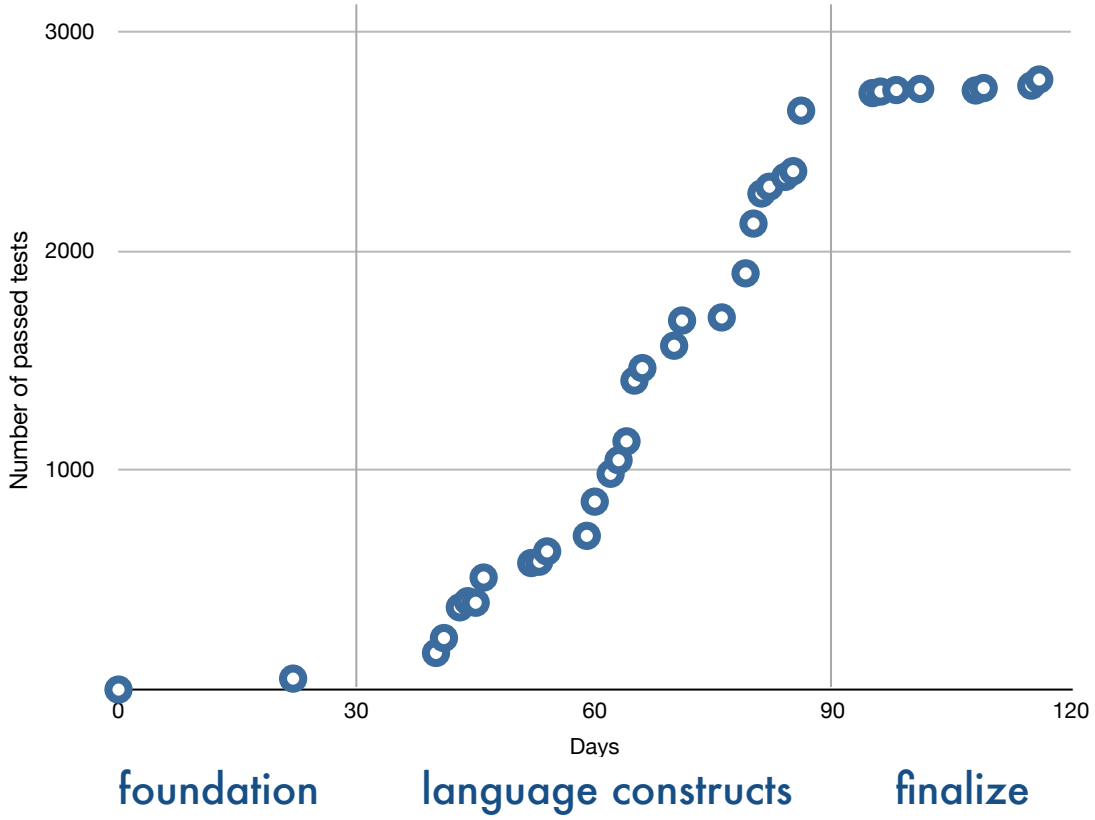
semantic rules: 1,370

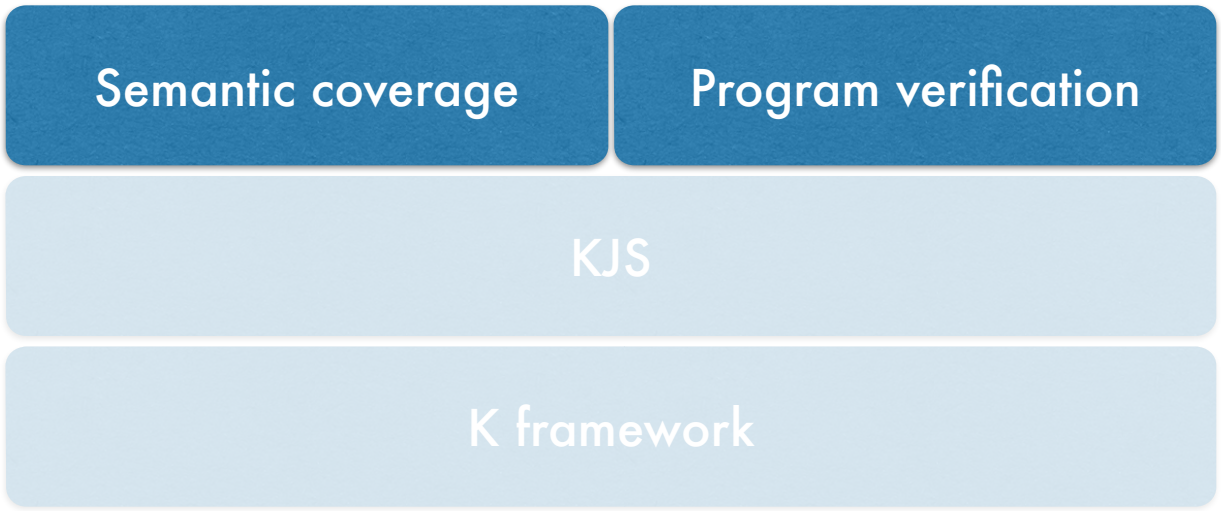


Development cost

Took only *four months* by a first year PhD student.

semantic rules: 1,370




Semantic coverage
Program verification
KJS
K framework

Semantic coverage measurement

Prior attempts found it difficult to measure semantic coverage:

From Sep 23 2014 ECMA Committee Meeting Minutes:
(https://esdiscuss.org/notes/2014-09-23)

Discussion of test262's (lack of) coverage.

test262 maintainer → Brian Terlson: *We didn't have coverage in test262 of …*

…

(Discussion of running test262 tests against implementations, especially given their optimizations.)

…

creator of JavaScript → Brendan Eich: *this was the "depress the room" agenda item*

…

Conclusion/Resolution

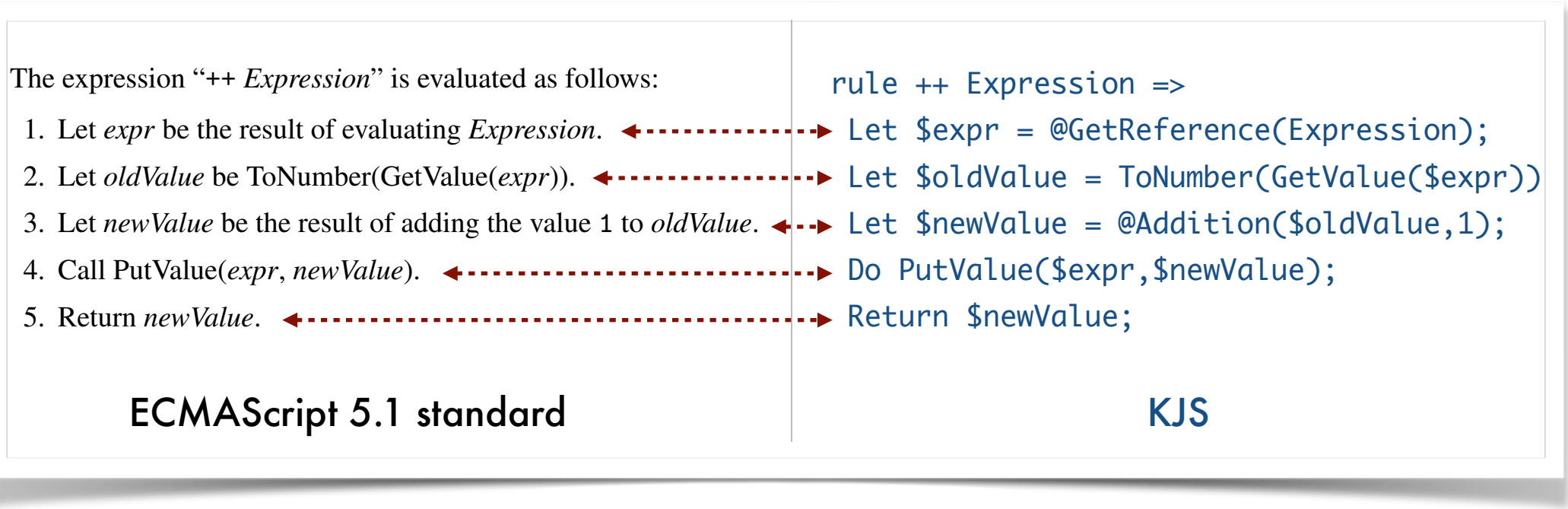
- It's impossible to test ECMAScript
- Testing is hard

Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?

Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?



Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?

Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?

17 rules *never* covered:

Page #	Section # - Step #	KJS	Po	Bo	CR	FF	SF
p35	8.7.1 GetValue (V) - [[Get]], Step 6	○	×	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.a	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.b	○	⊗	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.a	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.b	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 6.a & 6.b	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 7.a	○	×	○	○	×	○
p40	8.12.4 [[CanPut]] (P) - Step 8.a	○	⊗	⊗	○	○	○
p53	10.2.1.1.3 SetMutableBinding (N,V,S) - Step 4	○	×	○	×	○	×
p53	10.2.1.1.4 GetBindingValue(N,S) - Step 3.a	-	-	-	-	-	-
p53	10.2.1.1.5 DeleteBinding (N) - Step 2	-	-	-	-	-	-
p54	10.2.1.1.5 DeleteBinding (N) - Step 4 & 5	○	⊗	○	○	○	○
p55	10.2.1.2.4 GetBindingValue(N,S) - Step 4.a	-	-	-	-	-	-
p59	10.5 Declaration Binding Instantiation - Step 5.e.iii.1	○	○	○	○	○	○
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 1st condition is true	○	⊗	⊗	○	○	×
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 2nd condition is true	○	⊗	⊗	○	○	×
p62	10.6 Arguments Object - [[DefineOwnProperty]], Step 4.a, else-branch	-	-	-	-	-	-

○: Passed ×: Failed ⊗: Not applicable (failed due to unsupported semantics) -: Infeasible semantic behaviors

Po: [Politz et al. 2012] *Bo*: [Bodin et al. 2014] *CR*: Chrome 38.0 (V8 3.28.71) *FF*: Firefox 32.0 (SpiderMonkey 32) *SF*: Safari 7.0.4 (WebKit 537.76.4)

Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?

17 rules *never* covered:

- 6: shown infeasible → inconsistency of standard

Page #	Section # - Step #	KJS	Po	Bo	CR	FF	SF
p35	8.7.1 GetValue (V) - [[Get]], Step 6	○	×	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.a	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.b	○	⊗	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.a	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.b	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 6.a & 6.b	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 7.a	○	×	○	○	×	○
p40	8.12.4 [[CanPut]] (P) - Step 8.a	○	⊗	⊗	○	○	○
p53	10.2.1.1.3 SetMutableBinding (N,V,S) - Step 4	○	×	○	×	○	×
p53	10.2.1.1.4 GetBindingValue(N,S) - Step 3.a	-	-	-	-	-	-
p53	10.2.1.1.5 DeleteBinding (N) - Step 2	-	-	-	-	-	-
p54	10.2.1.1.5 DeleteBinding (N) - Step 4 & 5	○	⊗	○	○	○	○
p55	10.2.1.2.4 GetBindingValue(N,S) - Step 4.a	-	-	-	-	-	-
p59	10.5 Declaration Binding Instantiation - Step 5.e.iii.1	○	○	○	○	○	○
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 1st condition is true	○	⊗	⊗	○	○	×
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 2nd condition is true	○	⊗	⊗	○	○	×
p62	10.6 Arguments Object - [[DefineOwnProperty]], Step 4.a, else branch	-	-	-	-	-	-

○: Passed ×: Failed ⊗: Not applicable (failed due to unsupported semantics) -: Infeasible semantic behaviors

Po: [Politz et al. 2012] *Bo*: [Bodin et al. 2014] *CR*: Chrome 38.0 (V8 3.28.71) *FF*: Firefox 32.0 (SpiderMonkey 32) *SF*: Safari 7.0.4 (WebKit 537.76.4)

Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?

17 rules *never* covered:

- 6: shown infeasible → inconsistency of standard
- 11: wrote new tests → found bugs in JS engines

Page #	Section # - Step #	KJS	Po	Bo	CR	FF	SF
p35	8.7.1 GetValue (V) - [[Get]], Step 6	○		⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.a	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.b	○	⊗	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.a	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.b	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 6.a & 6.b	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 7.a	○		○	○		○
p40	8.12.4 [[CanPut]] (P) - Step 8.a	○	⊗	⊗	○	○	○
p53	10.2.1.1.3 SetMutableBinding (N,V,S) - Step 4	○		○		○	
p53	10.2.1.1.4 GetBindingValue(N,S) - Step 3.a	-	-	-	-	-	-
p53	10.2.1.1.5 DeleteBinding (N) - Step 2	-	-	-	-	-	-
p54	10.2.1.1.5 DeleteBinding (N) - Step 4 & 5	○	⊗	○	○	○	○
p55	10.2.1.2.4 GetBindingValue(N,S) - Step 4.a	-	-	-	-	-	-
p59	10.5 Declaration Binding Instantiation - Step 5.e.iii.1	○	○	○	○	○	
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 1st condition is true	○	⊗	⊗	○	○	
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 2nd condition is true	○	⊗	⊗	○	○	
p62	10.6 Arguments Object - [[DefineOwnProperty]], Step 4.a, else branch	-	-	-	-	-	-

○: Passed ×: Failed ⊗: Not applicable (failed due to unsupported semantics) -: Infeasible semantic behaviors

Po: [Politz et al. 2012] *Bo*: [Bodin et al. 2014] *CR*: Chrome 38.0 (V8 3.28.71) *FF*: Firefox 32.0 (SpiderMonkey 32) *SF*: Safari 7.0.4 (WebKit 537.76.4)

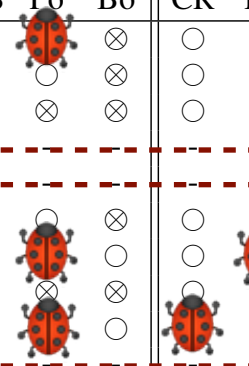
Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?

17 rules *never* covered:

- 6: shown infeasible → inconsistency of standard
- 11: wrote new tests → found bugs in JS engines

Page #	Section # - Step #	KJS	Po	Bo	CR	FF	SF
p35	8.7.1 GetValue (V) - [[Get]], Step 6	○		⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.a	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.b	○	⊗	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.a	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.b	-	-	-	-	-	-
p36	8.7.2 PutValue (V, W) - [[Put]], Step 6.a & 6.b	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 7.a	○		○	○		○
p40	8.12.4 [[CanPut]] (P) - Step 8.a	○	⊗	⊗	○	○	○
p53	10.2.1.1.3 SetMutableBinding (N,V,S) - Step 4	○		○		○	
p53	10.2.1.1.4 GetBindingValue(N,S) - Step 3.a	-	-	-	-	-	-
p53	10.2.1.1.5 DeleteBinding (N) - Step 2	-	-	-	-	-	-
p54	10.2.1.1.5 DeleteBinding (N) - Step 4 & 5	○	⊗	○	○	○	○
p55	10.2.1.2.4 GetBindingValue(N,S) - Step 4.a	-	-	-	-	-	-
p59	10.5 Declaration Binding Instantiation - Step 5.e.iii.1	○	○	○	○	○	
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 1st condition is true	○	⊗	⊗	○	○	
p59	10.5 Declaration Binding Instantiation - Step 5.e.iv, 2nd condition is true	○	⊗	⊗	○	○	
p62	10.6 Arguments Object - [[DefineOwnProperty]], Step 4.a, else-branch	-	-	-	-	-	-



motivating example

○: Passed ×: Failed ⊗: Not applicable (failed due to unsupported semantics) -: Infeasible semantic behaviors

Po: [Politz et al. 2012] *Bo*: [Bodin et al. 2014] *CR*: Chrome 38.0 (V8 3.28.71) *FF*: Firefox 32.0 (SpiderMonkey 32) *SF*: Safari 7.0.4 (WebKit 537.76.4)

Semantic coverage measurement

How many semantic rules are covered by 2,782 core tests?

17 rules *never* covered:

- 6: shown infeasible → inconsistency of standard
- 11: wrote new tests → found bugs in JS engines

Page #	Section # - Step #	KJS	Po	Bo	CR	FF	SF
p35	8.7.1 GetValue (V) - [[Get]], Step 6	○	🐞	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.a	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 2.b	○	⊗	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.a	–	–	–	–	–	–
p36	8.7.2 PutValue (V, W) - [[Put]], Step 4.b	–	–	–	–	–	–
p36	8.7.2 PutValue (V, W) - [[Put]], Step 6.a & 6.b	○	○	⊗	○	○	○
p36	8.7.2 PutValue (V, W) - [[Put]], Step 7.a	○	🐞	○	○	🐞	○
p40	8.12.4 [[CanPut]] (P) - Step 8.a	○	⊗	⊗	○	○	○
p53	10.2.1.1.3 SetMutableBinding (N,V,S) - Step 4	○	🐞	○	🐞	○	🐞 ← motivating example
p53	10.2.1.1.4 GetBindingValue(N,S) - Step 3.a	–	–	–	–	–	–
p53	10.2.1.1.5 DeleteBinding (N) - Step 2	–	–	–	–	–	–
p54	10.2.1.1.5 DeleteBinding (N) - Step 4 & 5	○	⊗	○	○	○	○
p55	10.2.1.2.4 GetBindingValue(N,S) - Step 4.a	–	–	–	–	–	–
p59	10.5 Declaration Binding Instantiation - Step 5.e.iii.1	○	○	○	○	○	🐞
p59	1(	○	⊗	⊗	○	○	🐞
p59	1(	○	⊗	⊗	○	○	🐞
p62	1(	–	–	–	–	–	–

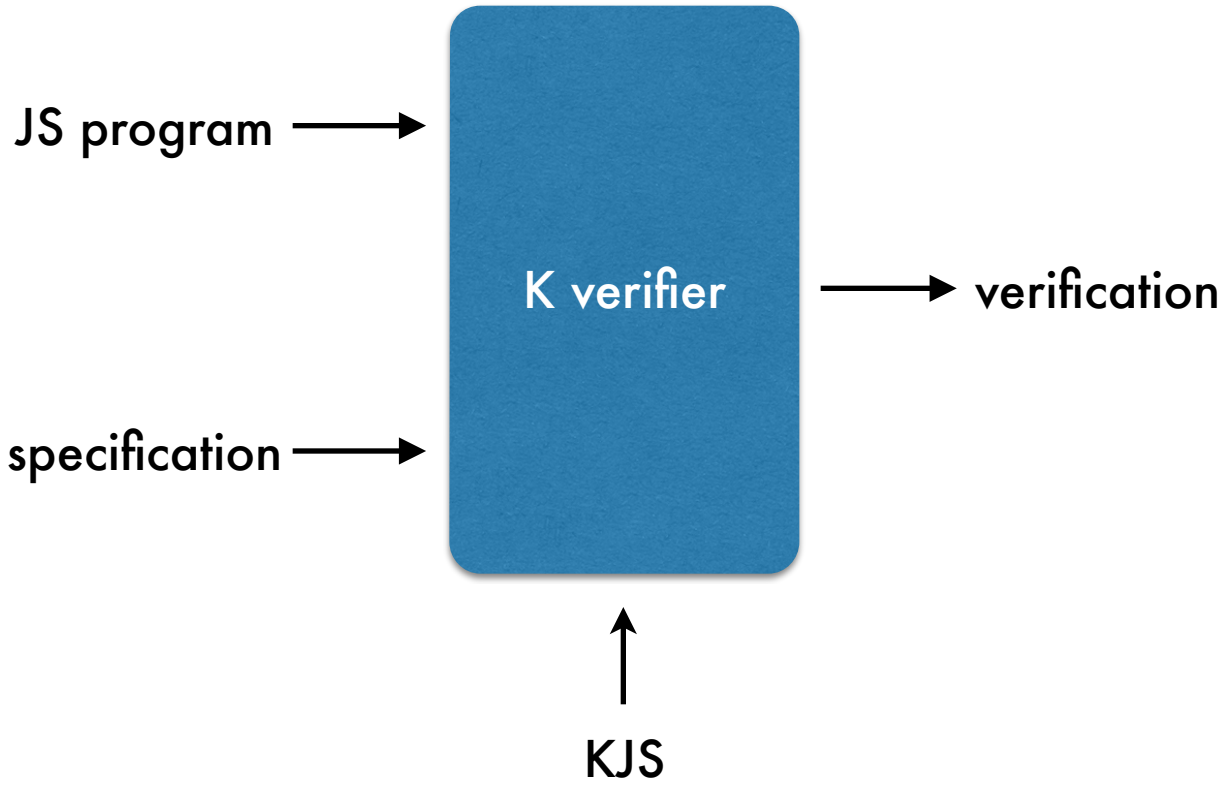
semantics is useful

○: Pas sible semantic behaviors

Po: [Politz et al. 2012] *B* ey 32) *SF*: Safari 7.0.4 (WebKit 537.76.4)

Program verification

Matching/Reachability Logic Verifier [Rosu and Stefanescu 2012, 2013, 2014]



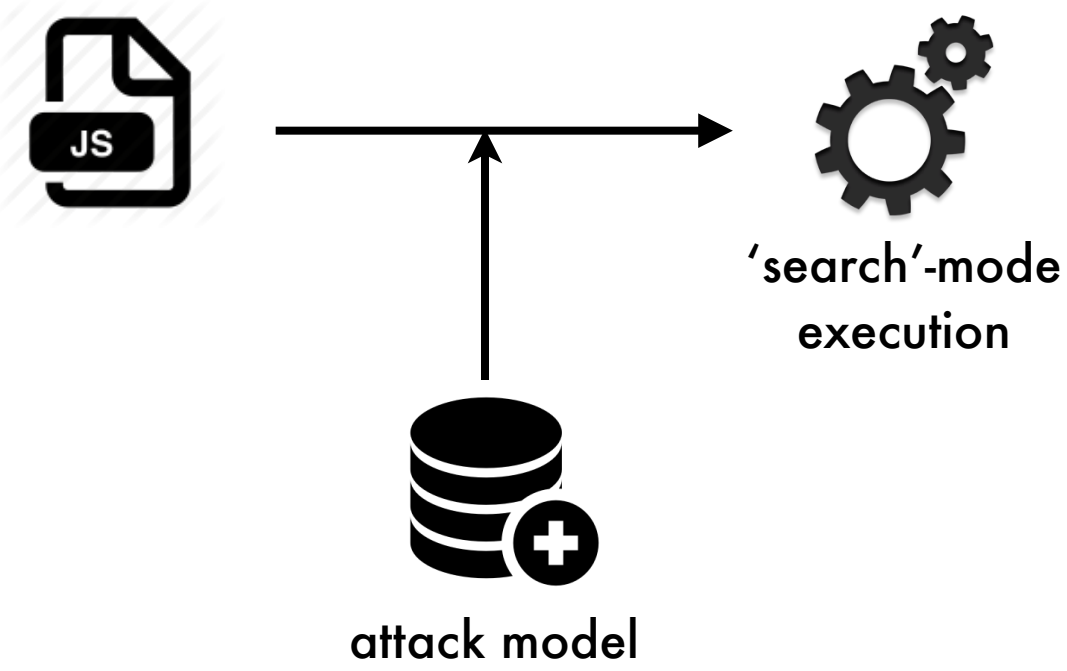
Program verification

Matching/Reachability Logic Verifier [Rosu and Stefanescu 2012, 2013, 2014]

Function	Size (LOC)	Time (s)
List reverse	13	8
List append	12	13
BST find	12	7
BST insert	23	12
BST delete	34	17
AVL find	11	7
AVL insert	87	109
AVL delete	106	174

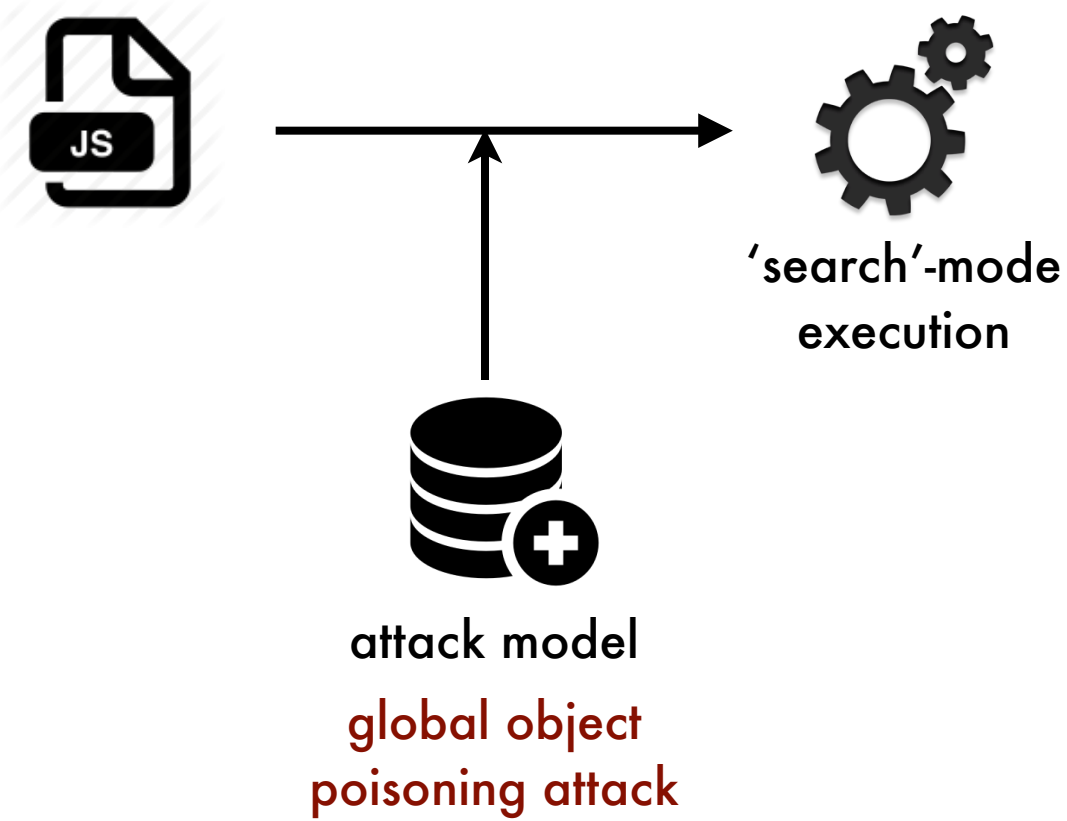
Security vulnerability detection

Found a known security vulnerability [Fournet et al. 2013]



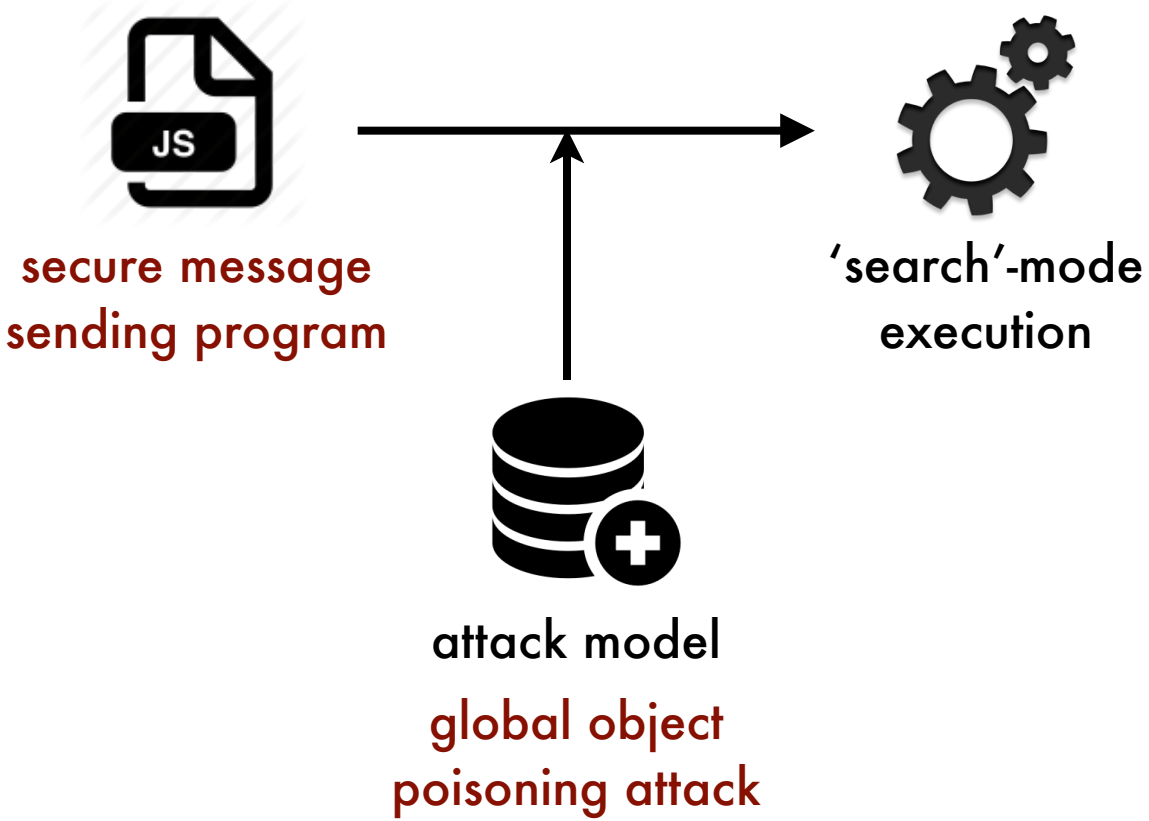
Security vulnerability detection

Found a known security vulnerability [Fournet et al. 2013]



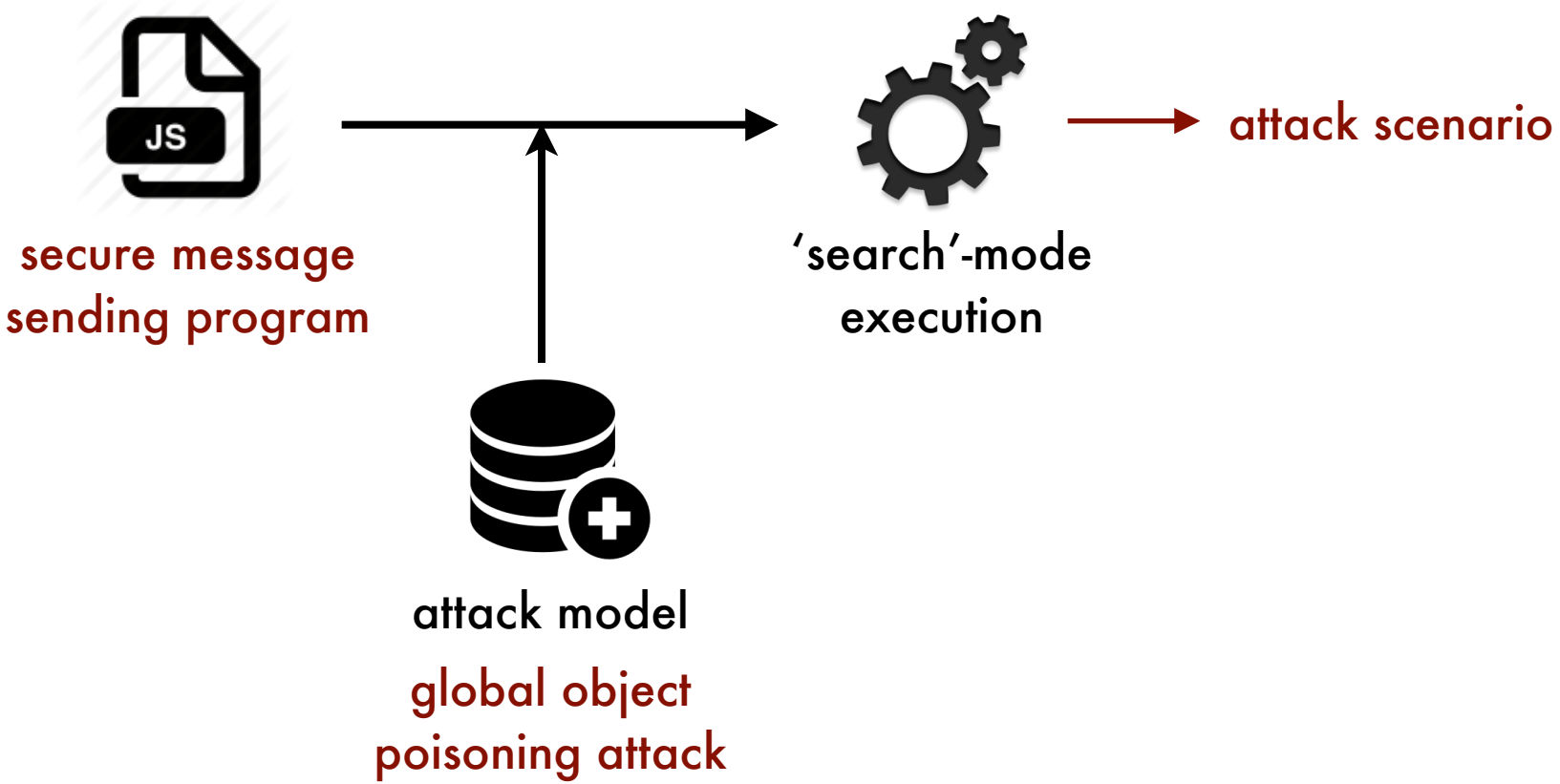
Security vulnerability detection

Found a known security vulnerability [Fournet et al. 2013]



Security vulnerability detection

Found a known security vulnerability [Fournet et al. 2013]



Semantic-driven formal analysis

Applications	Dev. time
semantic coverage measurement	1.5 weeks
program verification	2 weeks
security vulnerability detection	1 week

Summary

- Most complete JavaScript semantics to date.
- Semantic coverage measurement
  - Found bugs in Chrome, Firefox, and Safari
- Symbolically executable
  - Verified JavaScript programs
  - Found known security vulnerability

github.com/kframework/javascript-semantics