Formal JavaScript

Alan Schmitt

June 12, 2019

```
console.log((+[![]]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]
(!![]+[])[+!+[]]][([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!![]+
!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[
[]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]+(!![]+
+[]]+([][[]]+[])[+[]]+([][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[!+[]+!+[]]+(!![]+[])[+[]]+(!
]]+(!![]+[])[+!+[]]]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+[]]+(!![]+[][(![]+[])[+[]]+([![]]+[][[]])[+!+[]+[+[]]
+(!![]+[])[+[]]+(!![]+[])[!+[]+!+[]+!+[]]+(!![]+[])[+!+[]]])[+!+[]+[+[]]]+(!![]+[])[+!+[]]((!![]+[])[+!+
[]+!+[]]+(!![]+[])[+[]]+([][[]]+[])[+[]]+(!![]+[])[+!+[]]+([][[]]+[])[+!+[]]+(+[![]]+[][(![]+[])[+[]]+([!
```

Thank you for the invitation

Motivation

How do you trust your software?

How do you trust your software?

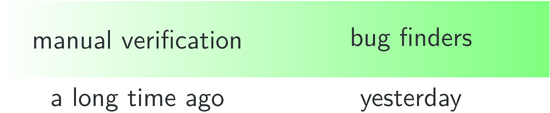
- Manual verification
  - does not scale

manual verification

a long time ago

How do you trust your software?

- Manual verification
  - does not scale
- Automatic bug finder
  - may miss some bugs



manual verification

bug finders

a long time ago

yesterday

How do you trust your software?

- Manual verification
  - does not scale
- Automatic bug finder
  - may miss some bugs
- Automatic, sound verifier
  - show the absence of bugs, may raise false alarms
    ex: the Astrée static analyzer

http://www.astree.ens.fr/

“Air France Airbus A380-800 F” by BriYYZ originally posted to Flickr as Arriving LAX north complex, credit BriYYZ, link

manual verification	bug finders	sound verifiers
a long time ago	yesterday	today

How do you trust the tool that verifies your software?

- Manual verification
  - does not scale
- Automatic bug finder
  - may miss some bugs
- Automatic, sound verifier
  - show the absence of bugs, may raise false alarms
    ex: the Astrée static analyzer

`http://www.astree.ens.fr/`

"Air France Airbus A380-800 F" by BriYYZ originally posted to Flickr as Arriving LAX north complex, credit BriYYZ, link

manual verification	bug finders	sound verifiers
a long time ago	yesterday	today

How do you trust the tool that verifies your software?

- Manual verification
  - does not scale
- Automatic bug finder
  - may miss some bugs
- Automatic, sound verifier
  - show the absence of bugs, may raise false alarms
    ex: the Astrée static analyzer
- Formally-verified verifier
  - the verifier comes with a soundness proof
    that is machine checked

http://www.astree.ens.fr/

"Air France Airbus A380-800 F" by BriYYZ originally posted to Flickr as Arriving LAX north complex, credit BriYYZ, link

manual verification	bug finders	sound verifiers	verified verifiers
a long time ago	yesterday	today	tomorrow

Goal

Certified Analyses for JavaScript

Why JavaScript?

1. JavaScript is everywhere
2. JavaScript matters for web security
3. JavaScript is complex
4. JavaScript comes with a specification

What is JavaScript?

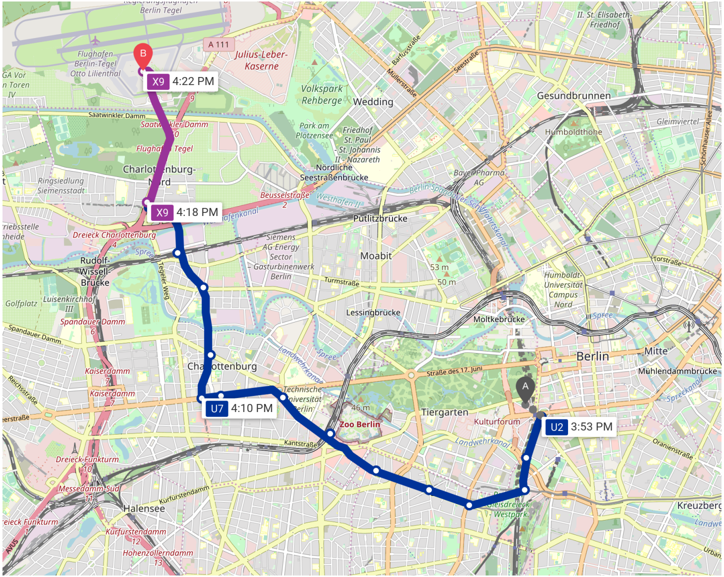
JavaScript Origin

Netscape Communications realized that the Web needed to become more dynamic. Marc Andreessen, the founder of the company believed that HTML needed a ”glue language” that was easy to use by Web designers and part-time programmers to assemble components such as images and plugins, where the code could be written directly in the Web page markup. – Wikipedia

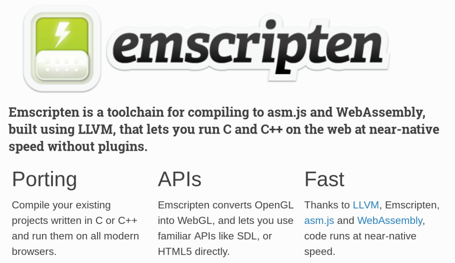
credit Brian Solis

Rich, Interactive Web Pages

diaspora*



JavaScript as Assembly of the Web



Emscripten is a toolchain for compiling to asm.js and WebAssembly, built using LLVM, that lets you run C and C++ on the web at near-native speed without plugins.

Porting

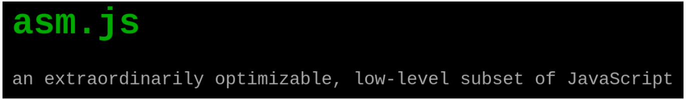
Compile your existing projects written in C or C++ and run them on all modern browsers.

APIs

Emscripten converts OpenGL into WebGL, and lets you use familiar APIs like SDL, or HTML5 directly.

Fast

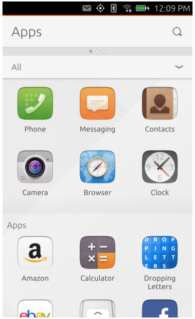
Thanks to LLVM, Emscripten, asm.js and WebAssembly, code runs at near-native speed.

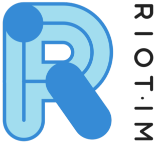


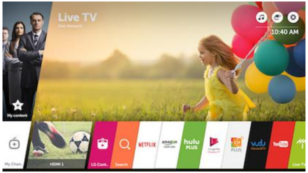
asm.js

an extraordinarily optimizable, low-level subset of JavaScript

JavaScript Beyond the Web

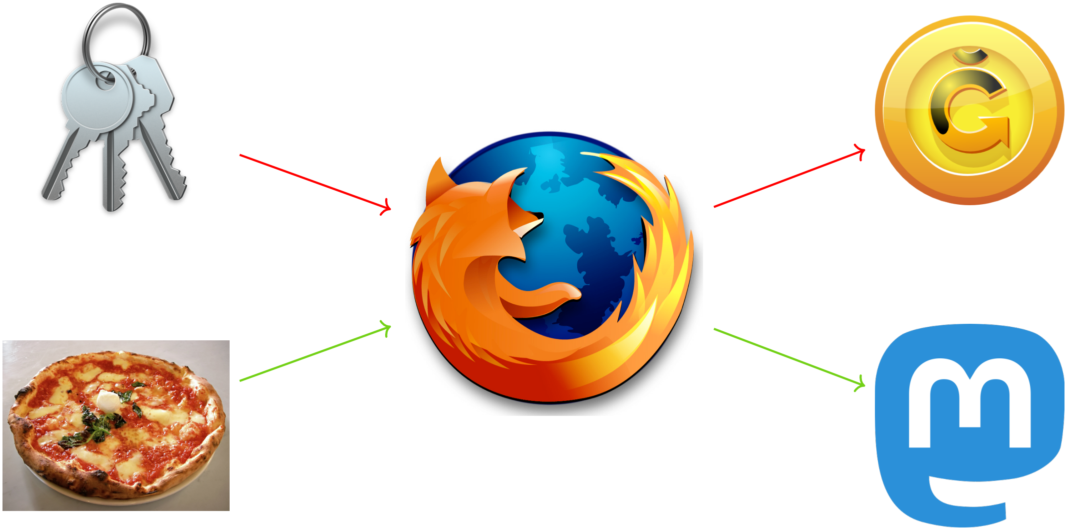






JavaScript and Web Security

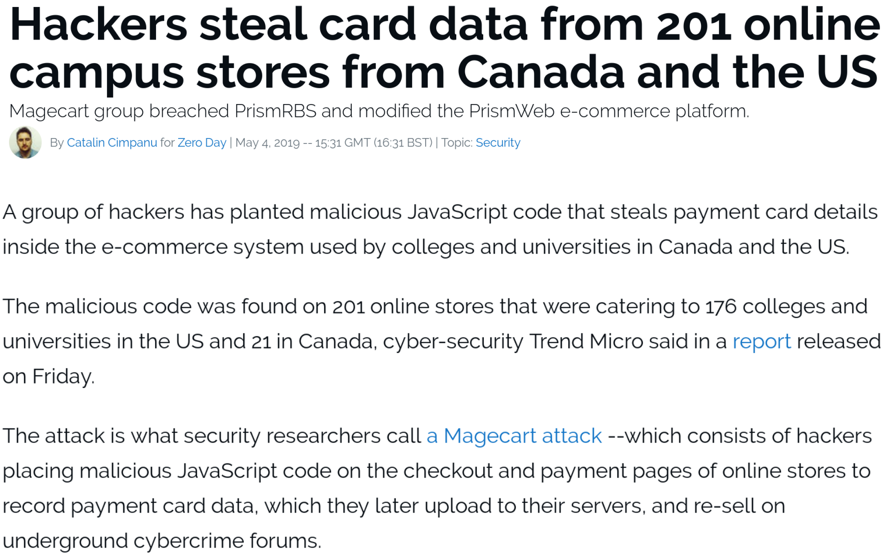
A Dangerous Situation



Mitigating Leaks

- sandbox (scripts cannot read or write files)
- same-origin policy (scripts cannot load scripts from somewhere else)

Mitigating Leaks



Hackers steal card data from 201 online campus stores from Canada and the US

Magecart group breached PrismRBS and modified the PrismWeb e-commerce platform.

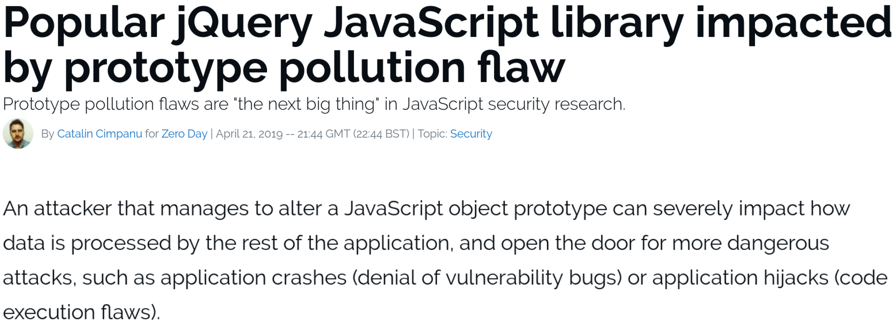
By Catalin Cimpanu for Zero Day | May 4, 2019 -- 15:31 GMT (16:31 BST) | Topic: Security

A group of hackers has planted malicious JavaScript code that steals payment card details inside the e-commerce system used by colleges and universities in Canada and the US.

The malicious code was found on 201 online stores that were catering to 176 colleges and universities in the US and 21 in Canada, cyber-security Trend Micro said in a report released on Friday.

The attack is what security researchers call a Magecart attack --which consists of hackers placing malicious JavaScript code on the checkout and payment pages of online stores to record payment card data, which they later upload to their servers, and re-sell on underground cybercrime forums.

Mitigating Leaks



Popular jQuery JavaScript library impacted by prototype pollution flaw

Prototype pollution flaws are "the next big thing" in JavaScript security research.

By Catalin Cimpanu for Zero Day | April 21, 2019 -- 21:44 GMT (22:44 BST) | Topic: Security

An attacker that manages to alter a JavaScript object prototype can severely impact how data is processed by the rest of the application, and open the door for more dangerous attacks, such as application crashes (denial of vulnerability bugs) or application hijacks (code execution flaws).

Javascript, the Language

Imperative and Functional

Variables

```
var x = 4
x = (10 * 4) + 2
console.log(x)
```

⇒ 42

Functions are Values

```
var f = function (g,x) {return (g(x) + 2)}

var fgx = f(function (y){return (10 * y)}, 4)

console.log("f(g,x) = " + fgx)
```

⇒ f(g,x) = 42

Objects

Literal Objects

```
var obj = { a : 1, b : 2 } /* littéral */
console.log (obj.a)         /* accès    */
```

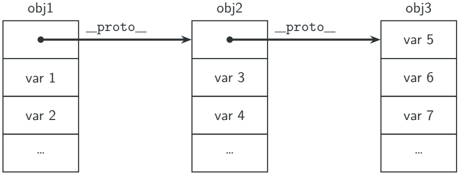
⇒ 1

Functions as Object Factories

```
function f(a) { this.x = a }
var o = new f(42)
console.log (o.x)
```

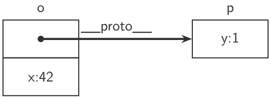
⇒ 42

A Language (Almost) Without Class



Prototypes and Object Factories

```
function f(a) { this.x = a }
var p = {y : 1}
f.prototype = p

var o = new f(42)

console.log("o.x = " + o.x + ", o.y = " + o.y)
```



⇒ `o.x = 42, o.y = 1`

Warning: the `prototype` field of a function becomes the `__proto__` field of the object created

A Language with (almost) no errors

Complex syntactic rules + automatic conversions ⇒ 🤪

Blocks vs Objects

```
var x = eval( "{} + {}" )
var y = eval("({} + {})")
console.log("x = " + x + "; y = " + y)
```

```
⇒ x = NaN; y = [object Object][object Object]
```

JSf*ck

```
var x = (![]+[])[+!+[]]
      + (![]+[])[!+[]+!+[]]
      + (![]+[])[+!+[]]
      + ([][[]]+[])[+!+[]]
console.log(x)
```

```
⇒ alan
```

A Language with (almost) no errors

Complex syntactic rules + automatic conversions ⇒ 🤪

Blocks vs Objects

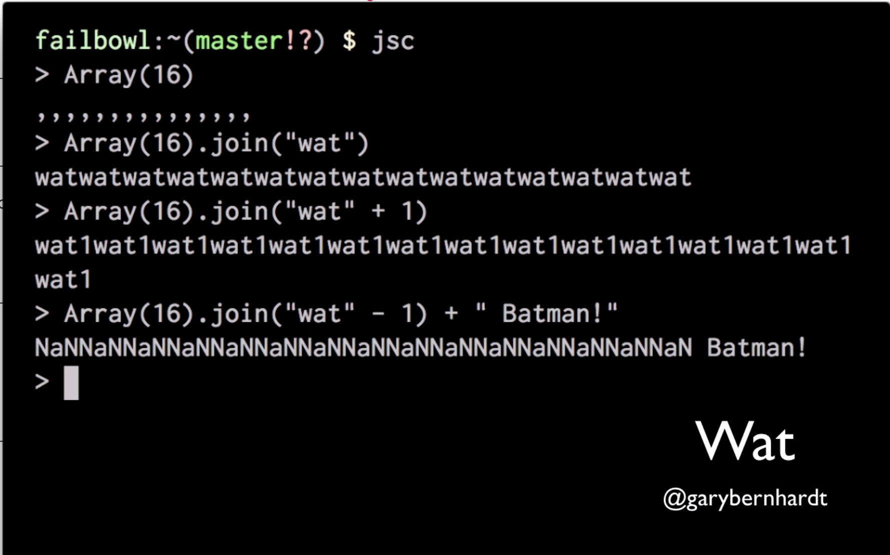
```
var x = eval( "{} + {}" )
var y = eval("({} + {})")
console.log("x = " + x + "
```

```
⇒ x = NaN; y = [object Ob
```

JSf*ck

```
var x = (![]+[])[+!+[]]
      + (![]+[])[!+[]+!+[]
      + (![]+[])[+!+[]]
      + ([][[]]+[])[+!+[]]
console.log(x)
```

```
⇒ alan
```



```
failbowl:~(master!?) $ jsc
> Array(16)

,,,,,,,,,,,,,,,
> Array(16).join("wat")
watwatwatwatwatwatwatwatwatwatwatwatwatwatwat
> Array(16).join("wat" + 1)
wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1wat1
wat1
> Array(16).join("wat" - 1) + " Batman!"
NaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaNNaN Batman!
> 
```

(![]+[])[+!+[]] is ’a’

`[]`	empty array	`[]`
`![]`	negation (converts to boolean)	`false` [1]
`![]+[]`	concatenation (converts to string)	`"false"`
`[]`	empty array	`[]`
`+[]`	conversion to number	`0`
`!+[]`	negation	`true`
`+!+[]`	conversion to number	`1`
`(![]+[])[+!+[]]`	array access	`'a'`

[1] Everything is `true` except `false`, 0, `NaN`, `""`, `null` and `undefined`

Conversion and User Code

```
var o = {}

o.toString = function () {
  o.toString = function () { return "😈" }
  return "😇"
}

console.log("I test                     : " + o)
console.log("It's all good, I can use it: " + o)
```

```
⇒ I test                     : 😇
  It's all good, I can use it: 😈
```

JavaScript and Web Browsers

Integration to Web Pages

- navigation

```
<input action="action" type="button" value="Back"
       onclick="history.go(-1);" />
```

- content modification (DOM)

```
document.title = "New title"
var para = document.createElement("p")
para.appendChild(document.createTextNode("Hello World!"))
document.getElementsByTagName("body")[0].appendChild(para)
```

The Event Loop

- JavaScript is a sequential language (for the moment)
- Pervasive use of *callbacks* and *asynchronous promises*

JavaScript, the Specification

A quick history of JavaScript and ECMAScript

1995 Brendan Eich hired by Netscape to embed Scheme

May 1995 as Java is included in Netscape, scripting should have a similar syntax; JavaScript prototype developed in 10 days

Dec. 1995 JavaScript deployed in Netscape Navigator 2.0 beta 3

Aug. 1996 JScript deployed in Internet Explorer 3.0

A quick history of JavaScript and ECMAScript

1995 Brendan Eich hired by Netscape to embed Scheme

May 1995 as Java is included in Netscape, scripting should have a similar syntax;
JavaScript prototype developed in 10 days

Dec. 1995 JavaScript deployed in Netscape Navigator 2.0 beta 3

Aug. 1996 JScript deployed in Internet Explorer 3.0

code is supposed to run identically in every browser

⇒ strong need for standardization

A quick history of JavaScript and ECMAScript

1995 Brendan Eich hired by Netscape to embed Scheme

May 1995 as Java is included in Netscape, scripting should have a similar syntax;
JavaScript prototype developed in 10 days

Dec. 1995 JavaScript deployed in Netscape Navigator 2.0 beta 3

Aug. 1996 JScript deployed in Internet Explorer 3.0

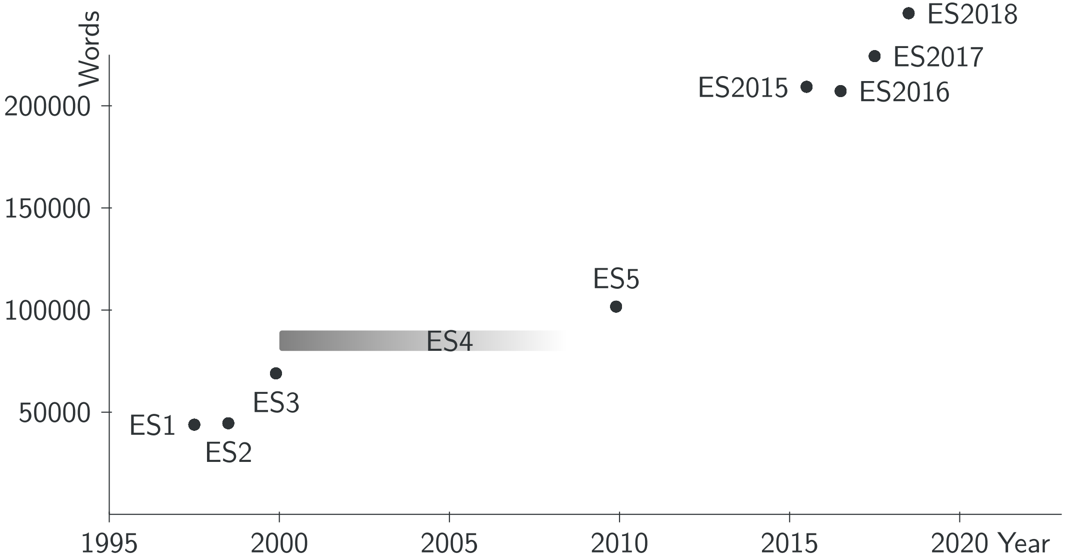
code is supposed to run identically in every browser

⇒ strong need for standardization

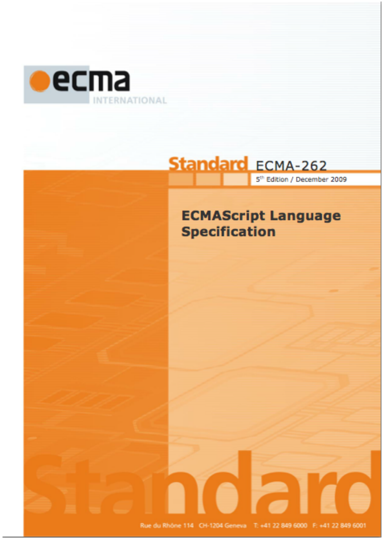
Nov. 1996 JavaScript submitted to Ecma International

June 1997 first edition of ECMA-262 (110 pages)

A quick history of JavaScript and ECMAScript



The specification



ecma INTERNATIONAL

Standard ECMA-262

5th Edition / December 2009

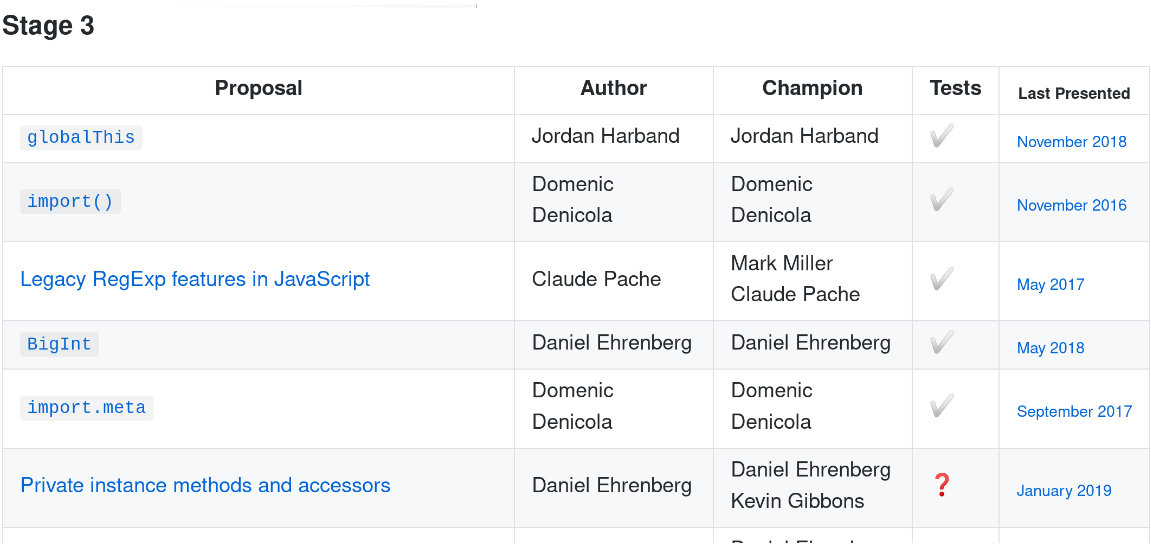
ECMAScript Language Specification

Standard

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

- new version every year
- 6 meetings of TC39 each year
- transparent process, on github

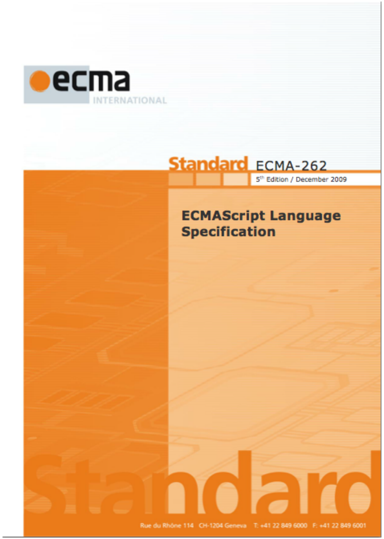
The specification



Stage 3

Proposal	Author	Champion	Tests	Last Presented
`globalThis`	Jordan Harband	Jordan Harband	✓	November 2018
`import()`	Domenic Denicola	Domenic Denicola	✓	November 2016
Legacy RegExp features in JavaScript	Claude Pache	Mark Miller Claude Pache	✓	May 2017
`BigInt`	Daniel Ehrenberg	Daniel Ehrenberg	✓	May 2018
`import.meta`	Domenic Denicola	Domenic Denicola	✓	September 2017
Private instance methods and accessors	Daniel Ehrenberg	Daniel Ehrenberg Kevin Gibbons	?	January 2019

The specification



ecma INTERNATIONAL

Standard ECMA-262

5th Edition / December 2009

ECMAScript Language Specification

Standard

Rue du Rhône 114 CH-1204 Geneva T: +41 22 849 6000 F: +41 22 849 6001

- new version every year
- 6 meetings of TC39 each year
- transparent process, on github
- don't break the web

Don’t Break the Web

Prototype Access and Mutation

```
function f() {}
f.prototype = { y : 2 }

var x1 = new f()
var x2 = new f()

console.log("Before: x1.y = " + x1.y + "; x2.y = " + x2.y)

x1.__proto__.y = 3

console.log("After:  x1.y = " + x1.y + "; x2.y = " + x2.y)
```

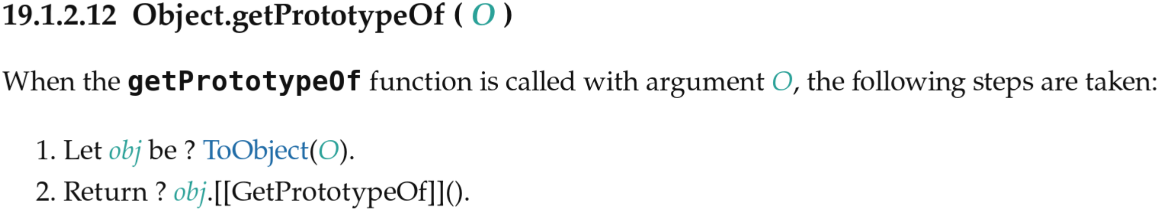
⇒

```
Before: x1.y = 2; x2.y = 2
After:  x1.y = 3; x2.y = 3
```

Don’t Break the Web

Prototype Access and Mutation

```
function f() {}
f.prototype = { y : 2 }

var x1 = new f()
var x2 = new f()

console.log("Before: x1.y = " + x1.y + "; x2.y = " + x2.y)
```

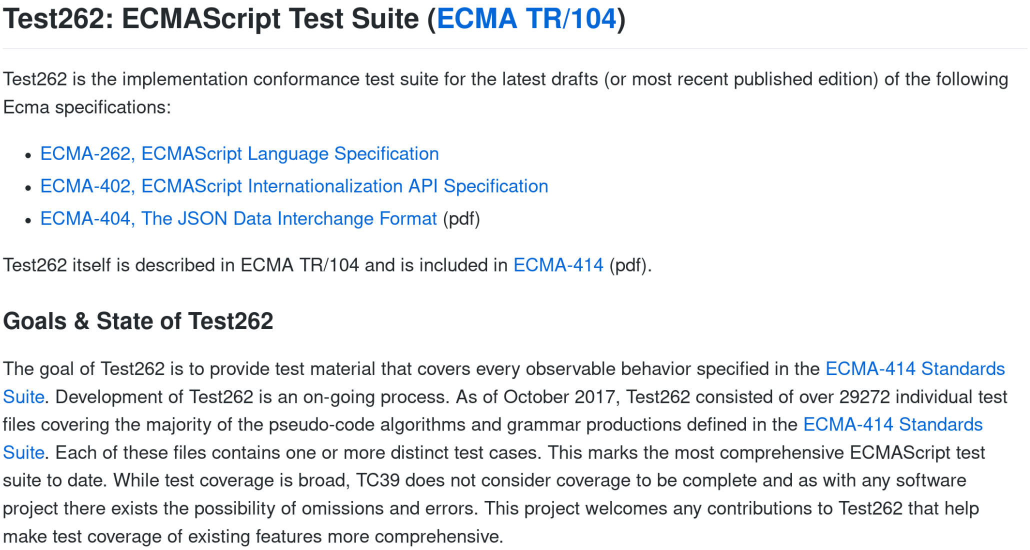


19.1.2.12 Object.getPrototypeOf (*O*)

When the **`getPrototypeOf`** function is called with argument *O*, the following steps are taken:

1. Let *obj* be ? ToObject(*O*).
2. Return ? *obj*.[[GetPrototypeOf]]().

Test 262



Test262: ECMAScript Test Suite (ECMA TR/104)

Test262 is the implementation conformance test suite for the latest drafts (or most recent published edition) of the following Ecma specifications:

- ECMA-262, ECMAScript Language Specification
- ECMA-402, ECMAScript Internationalization API Specification
- ECMA-404, The JSON Data Interchange Format (pdf)

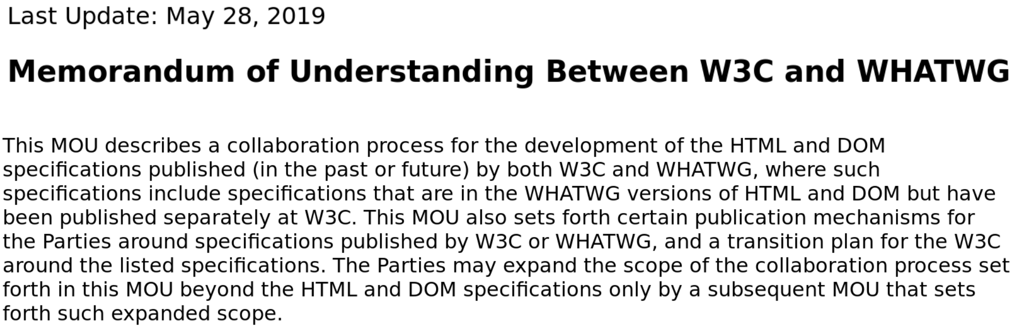
Test262 itself is described in ECMA TR/104 and is included in ECMA-414 (pdf).

Goals & State of Test262

The goal of Test262 is to provide test material that covers every observable behavior specified in the ECMA-414 Standards Suite. Development of Test262 is an on-going process. As of October 2017, Test262 consisted of over 29272 individual test files covering the majority of the pseudo-code algorithms and grammar productions defined in the ECMA-414 Standards Suite. Each of these files contains one or more distinct test cases. This marks the most comprehensive ECMAScript test suite to date. While test coverage is broad, TC39 does not consider coverage to be complete and as with any software project there exists the possibility of omissions and errors. This project welcomes any contributions to Test262 that help make test coverage of existing features more comprehensive.

TC39 and the Web

- scheduling (jobs, event loop)
- module loading
- DOM



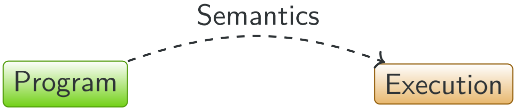
Last Update: May 28, 2019

Memorandum of Understanding Between W3C and WHATWG

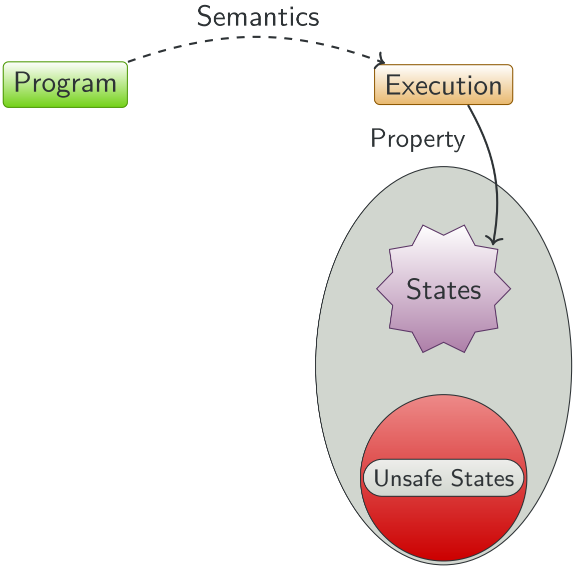
This MOU describes a collaboration process for the development of the HTML and DOM specifications published (in the past or future) by both W3C and WHATWG, where such specifications include specifications that are in the WHATWG versions of HTML and DOM but have been published separately at W3C. This MOU also sets forth certain publication mechanisms for the Parties around specifications published by W3C or WHATWG, and a transition plan for the W3C around the listed specifications. The Parties may expand the scope of the collaboration process set forth in this MOU beyond the HTML and DOM specifications only by a subsequent MOU that sets forth such expanded scope.

JavaScript, the Formalization

Certified Analyses



Certified Analyses



Semantics

Program

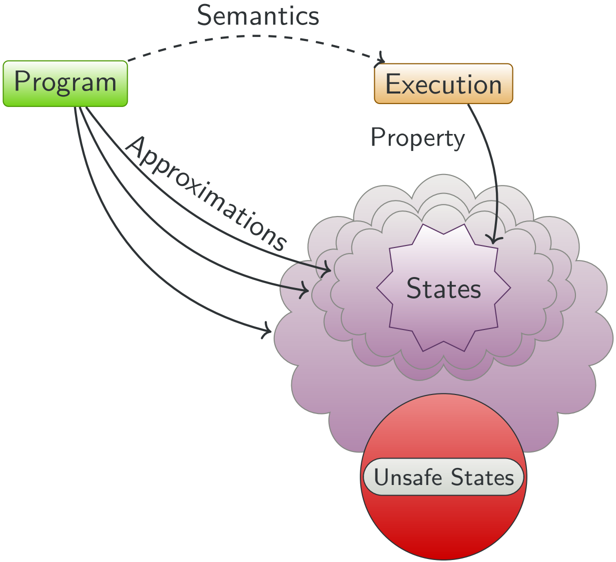
Execution

Property

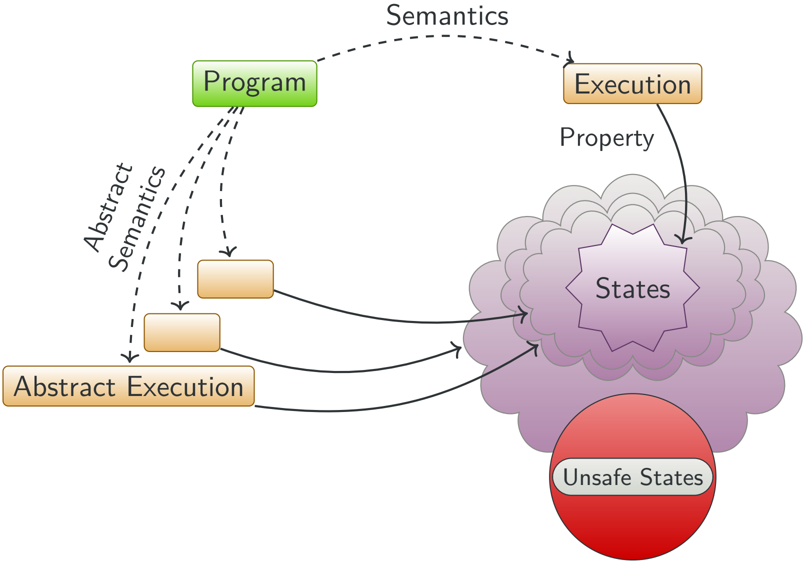
States

Unsafe States

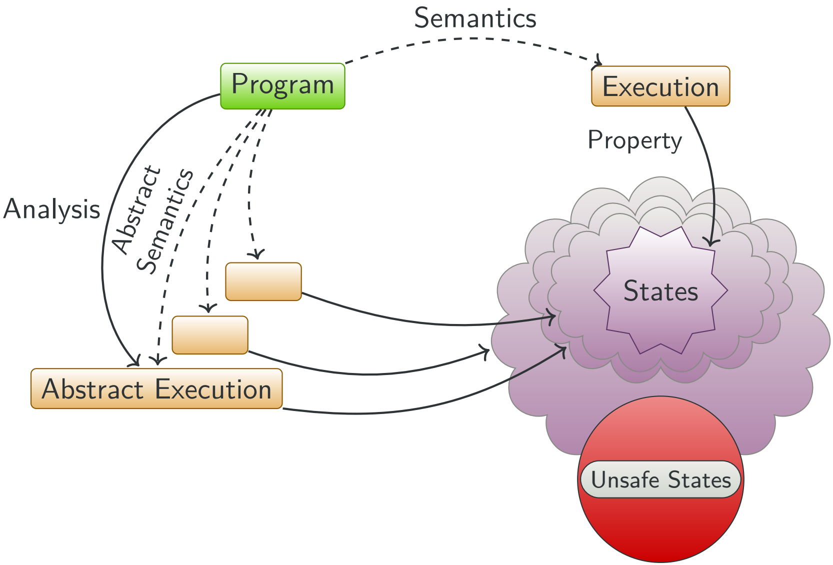
Certified Analyses



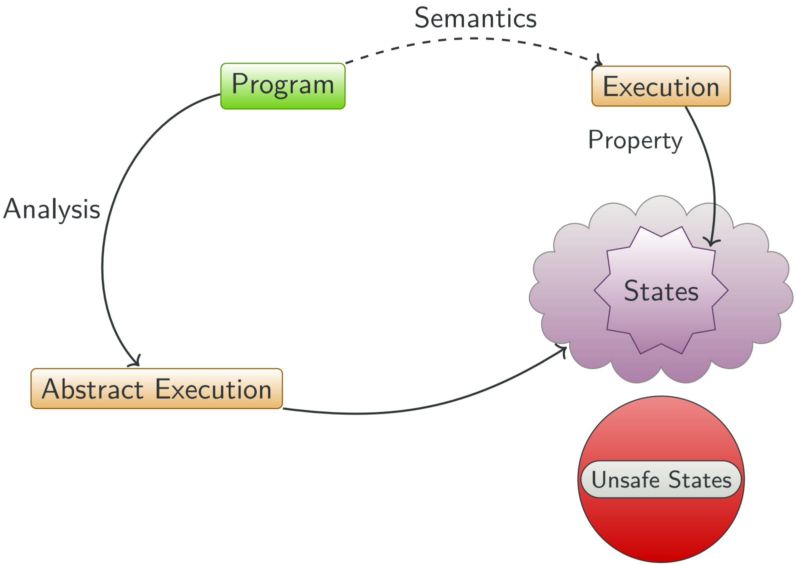
Certified Analyses



Certified Analyses



Certified Analyses



JSCert (2013)

Two JavaScript semantics in Coq

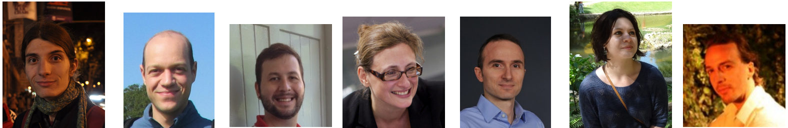
descriptive given a program and a result, say if they are related

executable given a program, compute the result

Correctness

If program P executes to v, then P and v are related

- 2 years, 8 people
- 18 klocs of Coq

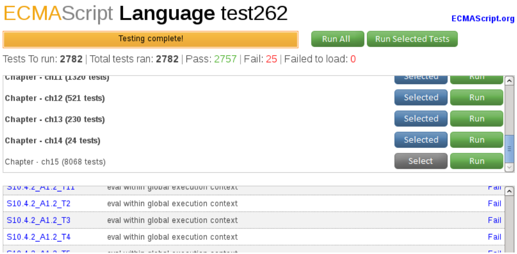


Certified JavaScript
JSCert
Careful, the beverage you're about to enjoy is extremely hot.

Did we Formalize JavaScript?

Stay close to the specification text

Test It


ECMAScript Language test262

ECMAScript.org

Testing complete!

Tests To run: 2782 | Total tests ran: 2782 | Pass: 2757 | Fail: 25 | Failed to load: 0

Chapter - ch11 (1320 tests)

Chapter - ch12 (521 tests)

Chapter - ch13 (230 tests)

Chapter - ch14 (24 tests)

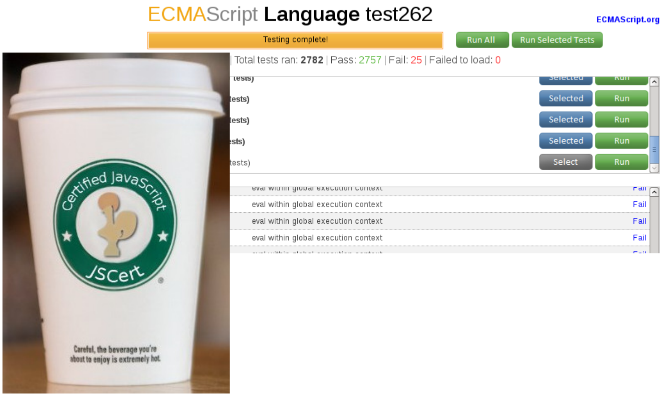
Chapter - ch15 (8068 tests)

S10.4.2_A1.2_T11	eval within global execution context	Fail
S10.4.2_A1.2_T2	eval within global execution context	Fail
S10.4.2_A1.2_T3	eval within global execution context	Fail
S10.4.2_A1.2_T4	eval within global execution context	Fail
S10.4.2_A1.2_T5	eval within global execution context	Fail

Did we Formalize JavaScript?

Stay close to the specification text

Test It


Certified JavaScript
JSCert
Careful, the beverage you're about to enjoy is extremely hot.

ECMAScript **Language** test262

ECMAScript.org

Testing complete!

Run All | Run Selected Tests

| Total tests ran: **2782** | Pass: 2757 | Fail: 25 | Failed to load: 0

tests) Selected Run

tests) Selected Run

tests) Selected Run

sts) Selected Run

tests) Select Run

eval within global execution context Fail

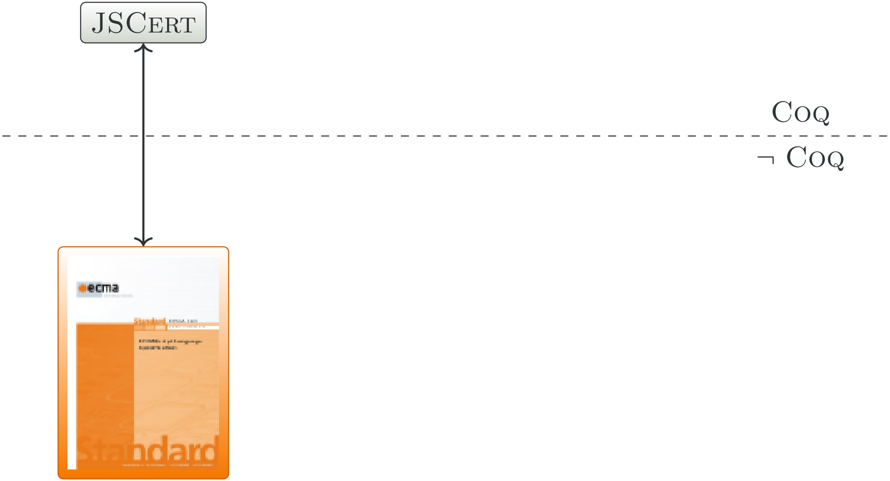
eval within global execution context Fail

eval within global execution context Fail

eval within global execution context Fail

eval within global execution context Fail

JavaScript Formalizations



JSCert

Coq

¬ Coq

Semantics of While

while (`Expression`) `Statement`

```
1. Let $V =$ empty.
2. Repeat
    1. Let *exprRef* be the result of evaluating Expression.
    2. If *ToBoolean*(*GetValue*(*exprRef*)) is false, return (*normal*, *V*, *empty*).
    3. Let *stmt* be the result of evaluating Statement.
    4. If *stmt.value* is not empty, let $V =$ *stmt.value*.
    5. If *stmt.type* is not continue or *stmt.target* is not in the current label set, then
        1. If *stmt.type* is break and *stmt.target* is in the current label set, then Return (*normal*, *V*, *empty*).
        2. If *stmt* is an abrupt completion, return *stmt*.
```

Semantics of While

1. Let $V =$ empty.

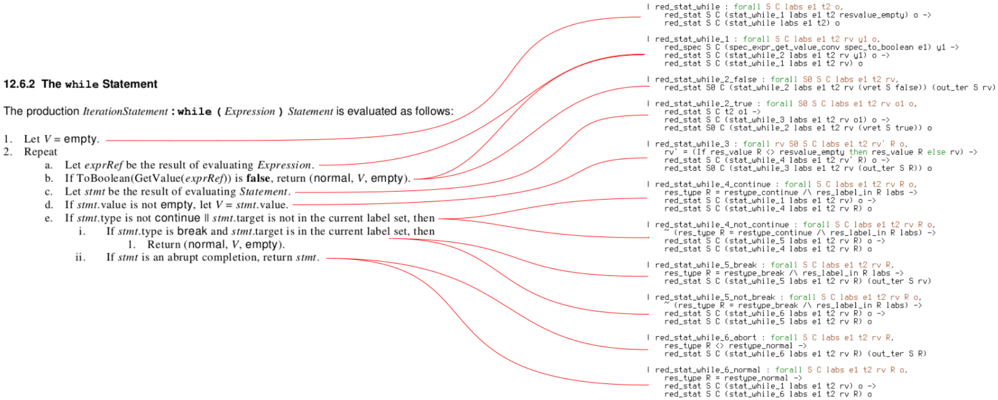
```
(* Step 1 *)
| red_stat_while : forall S C labs e1 t2 o,
    red_stat S C (stat_while_1 labs e1 t2 resvalue_empty) o ->
    red_stat S C (stat_while labs e1 t2) o
```

Semantics of While

2. Repeat
    1. Let *exprRef* be the result of evaluating `Expression`.
    2. If *ToBoolean*(*GetValue*(*exprRef*)) is `false`, return (*normal*, *V*, *empty*).

```
(* Steps 2a and 2b *)
| red_stat_while_1 : forall S C labs e1 t2 rv y1 o,
    red_spec S C (spec_expr_get_value_conv spec_to_boolean e1) y1 ->
    red_stat S C (stat_while_2 labs e1 t2 rv y1) o ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o

(* Step 2b False *)
| red_stat_while_2_false : forall S0 S C labs e1 t2 rv,
    red_stat S0 C (stat_while_2 labs e1 t2 rv (vret S false)) (out_ter S rv)
```

Close to the Specification



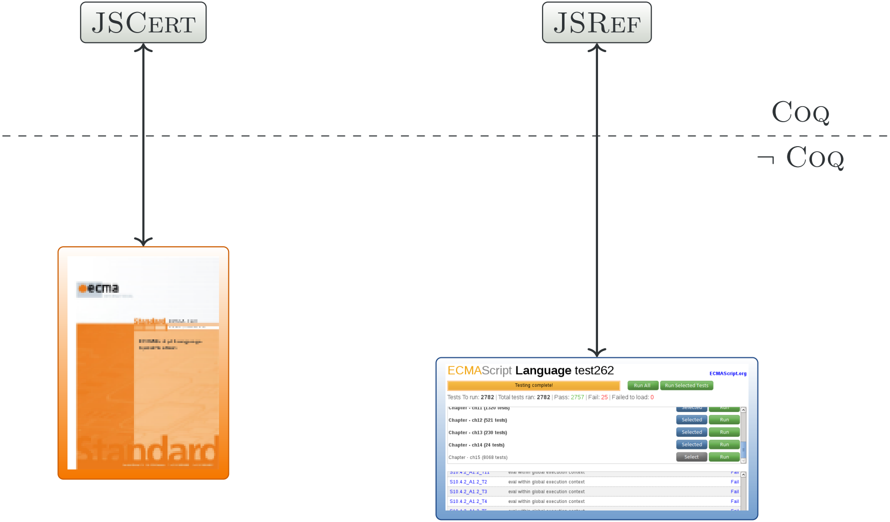
12.6.2 The `while` Statement

The production *IterationStatement* **: while (** *Expression* **)** *Statement* is evaluated as follows:

```
1.  Let V = empty.
2.  Repeat
    a.  Let exprRef be the result of evaluating Expression.
    b.  If ToBoolean(GetValue(exprRef)) is false, return (normal, V, empty).
    c.  Let stmt be the result of evaluating Statement.
    d.  If stmt.value is not empty, let V = stmt.value.
    e.  If stmt.type is not continue || stmt.target is not in the current label set, then
        i.   If stmt.type is break and stmt.target is in the current label set, then
             1.  Return (normal, V, empty).
        ii.  If stmt is an abrupt completion, return stmt.
```

```
| red_stat_while : forall S C labs e1 t2 o,
    red_stat S C (stat_while_1 labs e1 t2 resvalue_empty) o ->
    red_stat S C (stat_while labs e1 t2) o

| red_stat_while_1 : forall S C labs e1 t2 rv y1 o,
    red_spec S C (spec_expr_get_value_conv spec_to_boolean e1) y1 ->
    red_stat S C (stat_while_2 labs e1 t2 rv y1) o ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o

| red_stat_while_2_false : forall S0 S C labs e1 t2 rv,
    red_stat S0 C (stat_while_2 labs e1 t2 rv (vret S false)) (out_ter S rv)

| red_stat_while_2_true : forall S0 S C labs e1 t2 rv o1 o,
    red_stat S C t2 o1 ->
    red_stat S C (stat_while_3 labs e1 t2 rv o1) o ->
    red_stat S0 C (stat_while_2 labs e1 t2 rv (vret S true)) o

| red_stat_while_3 : forall rv S0 S C labs e1 t2 rv' R o,
    rv' = (If res_value R <> resvalue_empty then res_value R else rv) ->
    red_stat S C (stat_while_4 labs e1 t2 rv' R) o ->
    red_stat S0 C (stat_while_3 labs e1 t2 rv (out_ter S R)) o

| red_stat_while_4_continue : forall S C labs e1 t2 rv R o,
    res_type R = restype_continue /\ res_label_in R labs ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o ->
    red_stat S C (stat_while_4 labs e1 t2 rv R) o

| red_stat_while_4_not_continue : forall S C labs e1 t2 rv R o,
    ~ (res_type R = restype_continue /\ res_label_in R labs) ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) o ->
    red_stat S C (stat_while_4 labs e1 t2 rv R) o

| red_stat_while_5_break : forall S C labs e1 t2 rv R,
    res_type R = restype_break /\ res_label_in R labs ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) (out_ter S rv)

| red_stat_while_5_not_break : forall S C labs e1 t2 rv R o,
    ~ (res_type R = restype_break /\ res_label_in R labs) ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) o ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) o

| red_stat_while_6_abort : forall S C labs e1 t2 rv R,
    res_type R <> restype_normal ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) (out_ter S R)

| red_stat_while_6_normal : forall S C labs e1 t2 rv R o,
    res_type R = restype_normal ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) o
```

JavaScript Formalizations



ECMAScript **Language** test262

ECMAScript.org

Testing complete!

Tests To run: **2782** | Total tests ran: **2782** | Pass: 2757 | Fail: 25 | Failed to load: 0

Chapter - ch12 (521 tests) Selected Run

Chapter - ch13 (230 tests) Selected Run

Chapter - ch14 (24 tests) Selected Run

Chapter - ch15 (8068 tests) Select Run

```
S10.4.2_A1.2_T2    eval within global execution context    Fail
S10.4.2_A1.2_T3    eval within global execution context    Fail
S10.4.2_A1.2_T4    eval within global execution context    Fail
```

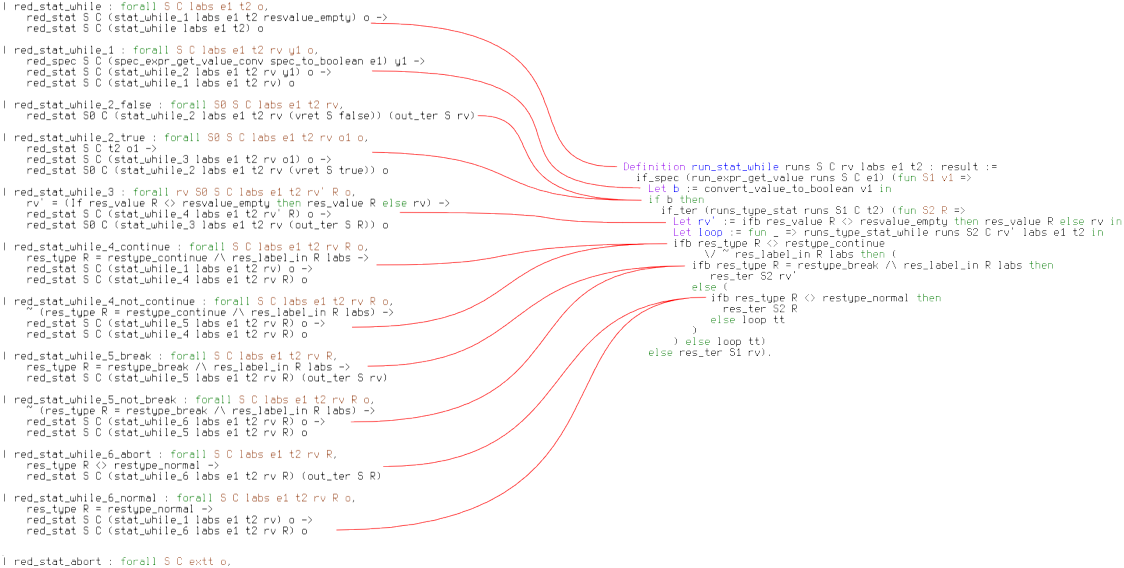
An Interpreter Written in Coq

```
Definition run_stat_while runs S C rv labs e1 t2 : result :=
 if_spec (run_expr_get_value runs S C e1) (fun S1 v1 =>
 Let b := convert_value_to_boolean v1 in
 if b then
  if_ter (runs_type_stat runs S1 C t2) (fun S2 R =>
  Let rv' := ifb res_value R <> resvalue_empty then res_value R else rv in
  Let loop := fun _ => runs_type_stat_while runs S2 C rv' labs e1 t2 in
  ifb res_type R <> restype_continue \/ ~ res_label_in R labs then (
   ifb res_type R = restype_break /\ res_label_in R labs then
     res_ter S2 rv'
   else (
    ifb res_type R <> restype_normal then
     res_ter S2 R
    else loop tt
   )
  ) else loop tt)
 else res_ter S1 rv).
```

Extracted to OCaml

```
let run_stat_while runs0 s c rv labs e1 t2 =
  if_spec (run_expr_get_value runs0 s c e1) (fun s1 v1 ->
    let_binding (convert_value_to_boolean v1) (fun b ->
      if b
      then if_ter (runs0.runs_type_stat s1 c t2) (fun s2 r ->
             let_binding
               (if not_decidable
                     (resvalue_comparable r.res_value Coq_resvalue_empty)
                then r.res_value
                else rv) (fun rv' ->
               let_binding (fun x ->
                 runs0.runs_type_stat_while s2 c rv' labs e1 t2) (fun loop ->
                 if or_decidable
                      (not_decidable (restype_comparable r.res_type Coq_restype_continue))
                      (not_decidable (bool_decidable (res_label_in r labs)))
                 then if and_decidable
                           (restype_comparable r.res_type Coq_restype_break)
                           (bool_decidable (res_label_in r labs))
                      then res_ter s2 (res_normal rv')
                      else if not_decidable (restype_comparable r.res_type Coq_restype_normal)
                           then res_ter s2 r
                           else loop ()
                 else loop ())))
      else res_ter s1 (res_normal rv)))
```

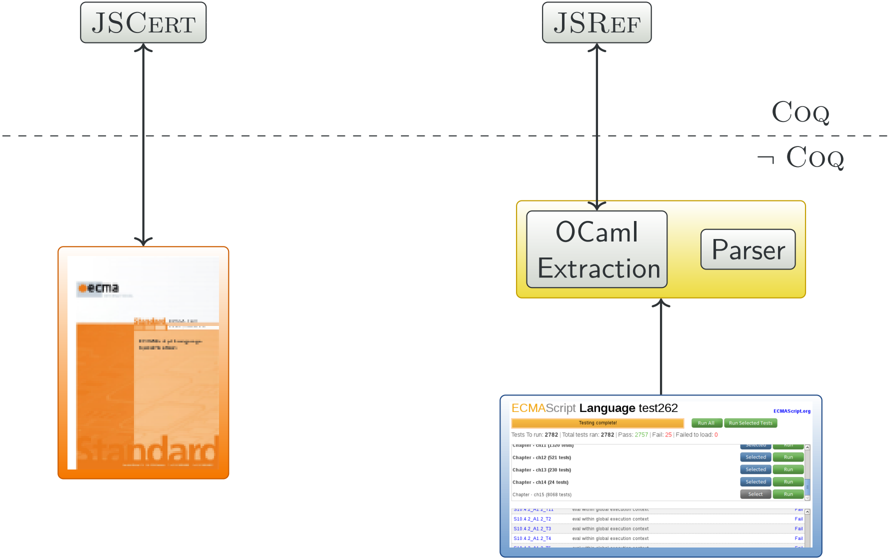
Fairly Close to the Inductive Rules



```
| red_stat_while : forall S C labs e1 t2 o,
    red_stat S C (stat_while_1 labs e1 t2 resvalue_empty) o ->
    red_stat S C (stat_while labs e1 t2) o

| red_stat_while_1 : forall S C labs e1 t2 rv y1 o,
    red_spec S C (spec_expr_get_value_conv spec_to_boolean e1) y1 ->
    red_stat S C (stat_while_2 labs e1 t2 rv y1) o ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o

| red_stat_while_2_false : forall S0 S C labs e1 t2 rv,
    red_stat S0 C (stat_while_2 labs e1 t2 rv (vret S false)) (out_ter S rv)

| red_stat_while_2_true : forall S0 S C labs e1 t2 rv o1 o,
    red_stat S C t2 o1 ->
    red_stat S C (stat_while_3 labs e1 t2 rv o1) o ->
    red_stat S0 C (stat_while_2 labs e1 t2 rv (vret S true)) o

| red_stat_while_3 : forall rv S0 S C labs e1 t2 rv' R o,
    rv' = (If res_value R <> resvalue_empty then res_value R else rv) ->
    red_stat S C (stat_while_4 labs e1 t2 rv' R) o ->
    red_stat S0 C (stat_while_3 labs e1 t2 rv (out_ter S R)) o

| red_stat_while_4_continue : forall S C labs e1 t2 rv R o,
    res_type R = restype_continue /\ res_label_in R labs ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o ->
    red_stat S C (stat_while_4 labs e1 t2 rv R) o

| red_stat_while_4_not_continue : forall S C labs e1 t2 rv R o,
    ~ (res_type R = restype_continue /\ res_label_in R labs) ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) o ->
    red_stat S C (stat_while_4 labs e1 t2 rv R) o

| red_stat_while_5_break : forall S C labs e1 t2 rv R,
    res_type R = restype_break /\ res_label_in R labs ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) (out_ter S rv)

| red_stat_while_5_not_break : forall S C labs e1 t2 rv R o,
    ~ (res_type R = restype_break /\ res_label_in R labs) ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) o ->
    red_stat S C (stat_while_5 labs e1 t2 rv R) o

| red_stat_while_6_abort : forall S C labs e1 t2 rv R,
    res_type R <> restype_normal ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) (out_ter S R)

| red_stat_while_6_normal : forall S C labs e1 t2 rv R o,
    res_type R = restype_normal ->
    red_stat S C (stat_while_1 labs e1 t2 rv) o ->
    red_stat S C (stat_while_6 labs e1 t2 rv R) o

| red_stat_abort : forall S C extt o,
```

```
Definition run_stat_while runs S C rv labs e1 t2 : result :=
  if_spec (run_expr_get_value runs S C e1) (fun S1 v1 =>
   Let b := convert_value_to_boolean v1 in
   if b then
     if_ter (runs_type_stat runs S1 C t2) (fun S2 R =>
       Let rv' := ifb res_value R <> resvalue_empty then res_value R else rv in
       Let loop := fun _ => runs_type_stat_while runs S2 C rv' labs e1 t2 in
       ifb res_type R <> restype_continue
           \/ ~ res_label_in R labs then (
         ifb res_type R = restype_break /\ res_label_in R labs then
           res_ter S2 rv'
         else (
           ifb res_type R <> restype_normal then
             res_ter S2 R
           else loop tt
         )
       ) else loop tt)
   else res_ter S1 rv).
```

JavaScript Formalizations



ECMAScript **Language** test262

Tests To run: **2782** | Total tests ran: **2782** | Pass: 2757 | Fail: 25 | Failed to load: 0

Chapter - ch11 (1320 tests)

Chapter - ch12 (521 tests)

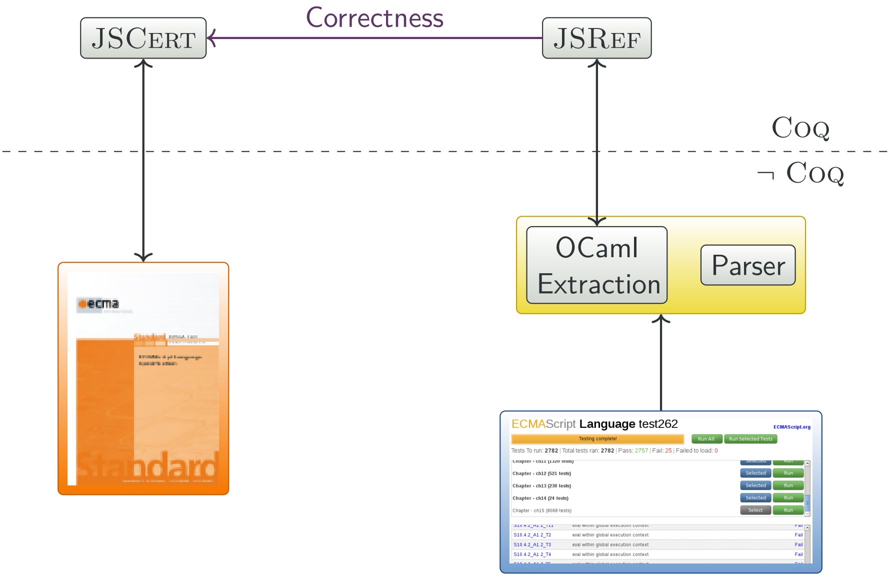
Chapter - ch13 (230 tests)

Chapter - ch14 (24 tests)

Chapter - ch15 (8068 tests)

```
S10.4.2_A1.2_T11    eval within global execution context    Fail
S10.4.2_A1.2_T2     eval within global execution context    Fail
S10.4.2_A1.2_T3     eval within global execution context    Fail
S10.4.2_A1.2_T4     eval within global execution context    Fail
```

JavaScript Formalizations



Correctness

JSCert

JSRef

Coq

¬ Coq

OCaml Extraction

Parser

ECMAScript Language test262

Tests To run: **2782** | Total tests ran: **2782** | Pass: 2757 | Fail: 25 | Failed to load: 0

Chapter - ch11 (1320 tests)

Chapter - ch12 (521 tests)

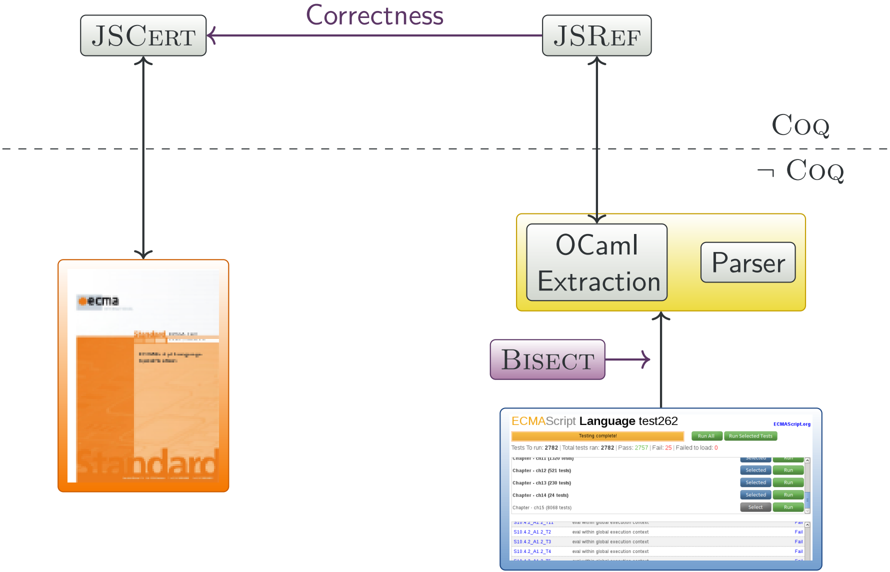
Chapter - ch13 (230 tests)

Chapter - ch14 (24 tests)

Chapter - ch15 (8068 tests)

S10.4.2_A1.2_T11	eval within global execution context	Fail
S10.4.2_A1.2_T2	eval within global execution context	Fail
S10.4.2_A1.2_T3	eval within global execution context	Fail
S10.4.2_A1.2_T4	eval within global execution context	Fail

JavaScript Formalizations



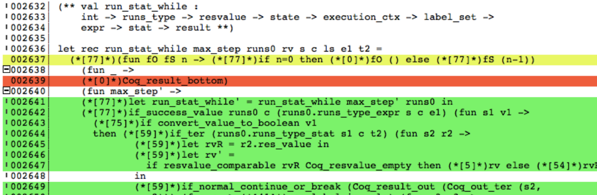
ECMAScript **Language** test262 ECMAScript.org

Tests To run: **2782** | Total tests ran: **2782** | Pass: 2757 | Fail: 25 | Failed to load: 0

```
Chapter - ch11 (1320 tests)        Selected  Run
Chapter - ch12 (521 tests)         Selected  Run
Chapter - ch13 (230 tests)         Selected  Run
Chapter - ch14 (24 tests)          Selected  Run
Chapter - ch15 (8068 tests)        Select    Run

S10.4.2_A1.2_T11   eval within global execution context   Fail
S10.4.2_A1.2_T2    eval within global execution context   Fail
S10.4.2_A1.2_T3    eval within global execution context   Fail
S10.4.2_A1.2_T4    eval within global execution context   Fail
S10.4.2_A1.2_T5    eval within global execution context   Fail
```

Test Suite Coverage with Bisect

- good coverage of the core of ECMAScript 5.1
- code extraction from JSRef
  1. instrumented to report coverage
  2. run the test suite
  3. find places not executed (not tested)
  4. relate to parts of the spec not tested
  5. discover discrepancies between implementations



```
(** val run_stat_while :
    int -> runs_type -> resvalue -> state -> execution_ctx -> label_set ->
    expr -> stat -> result **)

let rec run_stat_while max_step runs0 rv s c ls e1 t2 =
  (*[77]*)(fun fO fS n -> (*[77]*)if n=0 then (*[0]*)fO () else (*[77]*)fS (n-1))
    (fun _ ->
    (*[0]*)Coq_result_bottom)
    (fun max_step' ->
    (*[77]*)let run_stat_while' = run_stat_while max_step' runs0 in
    (*[77]*)if_success_value runs0 c (runs0.runs_type_expr s c e1) (fun s1 v1 ->
      (*[75]*)if convert_value_to_boolean v1
      then (*[59]*)if_ter (runs0.runs_type_stat s1 c t2) (fun s2 r2 ->
             (*[59]*)let rvR = r2.res_value in
             (*[59]*)let rv' =
               if resvalue_comparable rvR Coq_resvalue_empty then (*[5]*)rv else (*[54]*)rvR
             in
             (*[59]*)if_normal_continue_or_break (Coq_result_out (Coq_out_ter (s2,
```

`try` ... `finally` in Chrome

```
while(true) {
  try {
    "try" ;
    break
  } finally {
    "finally"
  }
}
```

returns `finally`

```
while(true) {
  try {
    "try" ;
    break
  } finally {
    "finally"
  };
  y = "done"
}
```

returns `try`

Impact on the Specification

- More precise definitions

```
1. Let lprim be ? ToPrimitive(lval).
2. Let rprim be ? ToPrimitive(rval).
3. If Type(lprim) is String or Type(rprim) is String, then
    a. Let lstr be ? ToString(lprim).
    b. Let rstr be ? ToString(rprim).
    c. Return the string-concatenation of lstr and rstr.
```

- Towards a typed specification
    - Numbers vs mathematical integers
    - Return values

JSExplain

An OCaml interpreter of JavaScript

- very close to the specification
- based on the extraction from JSRef
- uses a tiny subset of OCaml in monadic style
  - functions, tuples, shallow pattern matching, records
- request by Shu-yu Guo (Dagstuhl, 2014): a step by step execution of the spec

Specification

```
1. Let lprim be ? ToPrimitive(lval).
2. Let rprim be ? ToPrimitive(rval).
3. If Type(lprim) is String or Type(rprim) is String, then
    a. Let lstr be ? ToString(lprim).
    b. Let rstr be ? ToString(rprim).
    c. Return the string-concatenation of lstr and rstr.
4. Let lnum be ? ToNumber(lprim).
5. Let rnum be ? ToNumber(rprim).
6. Return the result of applying the addition operation to lnum and rnum.
```

Code

```
and run_binary_op_add s0 c v1 v2 =
  let%prim (s1, w1) = to_primitive_def s0 c v1 in
  let%prim (s2, w2) = to_primitive_def s1 c v2 in
  if  (type_compare (type_of (Value_prim w1)) Type_string)
   || (type_compare (type_of (Value_prim w2)) Type_string)
  then
    let%string (s3, str1) = to_string s2 c (Value_prim w1) in
    let%string (s4, str2) = to_string s3 c (Value_prim w2) in
    res_out (Out_ter (s4, (Res_val (Value_prim (Prim_string (strappend str1 str2)))))
  else
    let%number (s3, n1) = to_number s2 c (Value_prim w1) in
    let%number (s4, n2) = to_number s3 c (Value_prim w2) in
    res_out (Out_ter (s4, (Res_val (Value_prim (Prim_number (n1 +. n2))))))
```

Compiled to JavaScript

- motivation: run it in a browser
- uses compiler-libs to generate a typed AST, which we translate
- target is a tiny subset of JS
  - functions, objects (no prototype), arrays, string, numbers
  - no type conversion

```
var run_binary_op_add = function (s0, c, v1, v2) {
  return (if_prim(to_primitive_def(s0, c, v1), function(s1, w1) {
    return (if_prim(to_primitive_def(s1, c, v2), function(s2, w2) {
      if ((type_compare(type_of(Coq_value_prim(w1)), Coq_type_string())
        || type_compare(type_of(Coq_value_prim(w2)), Coq_type_string()))) {
          return (if_string(to_string(s2, c, Coq_value_prim(w1)), function(s3, str1) {
            return (if_string(to_string(s3, c, Coq_value_prim(w2)), function(s4, str2) {
              return (res_out(Coq_out_ter(s4, res_val(
                       Coq_value_prim(Coq_prim_string(strappend(str1, str2))))))); }));}));
      } else { ... }})); }));
};
```

and to Pseudo JavaScript

- to be readable while staying close to JavaScript
  - hide state and context
  - monadic extension of var
  - pattern matching
  - hide type changes

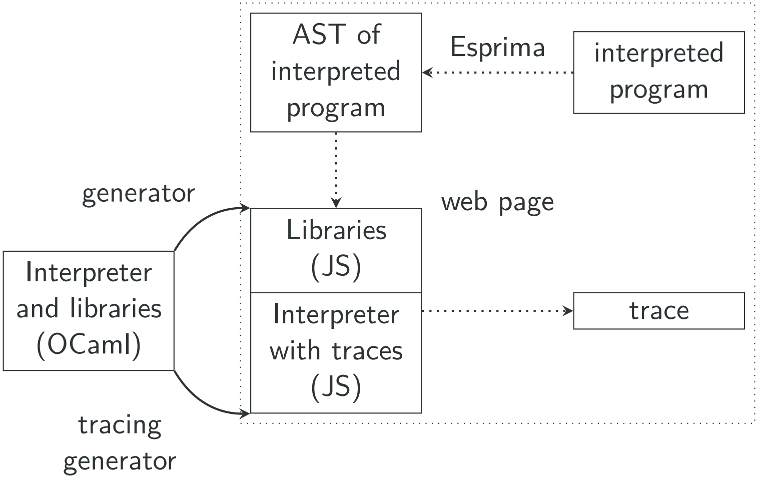
```
var run_expr_binary_op =
 function (op, e1, e2) {
  switch (op) {
   case Coq_binary_op_and:
     return (run_binary_op_and(e1, e2));
   case Coq_binary_op_or:
     return (run_binary_op_or(e1, e2));
   default:
     var%run v1 = run_expr_get_value(e1);
     var%run v2 = run_expr_get_value(e2);
     return (run_binary_op(op, v1, v2));
  }
};
```

```
var run_binary_op_add = function (v1, v2) {
 var%prim w1 = to_primitive_def v1;
 var%prim w2 = to_primitive_def v2;
 if ((type_cmp(type_of(w1), Type_string)
   || type_cmp(type_of(w2), Type_string))) {
   var%string str1 = to_string w1;
   var%string str2 = to_string w2;
   return (str_app(str1, str2));
 } else {
   var%number n1 = to_number w1;
   var%number n2 = to_number w2;
   return (n1 + n2);
 }
};
```

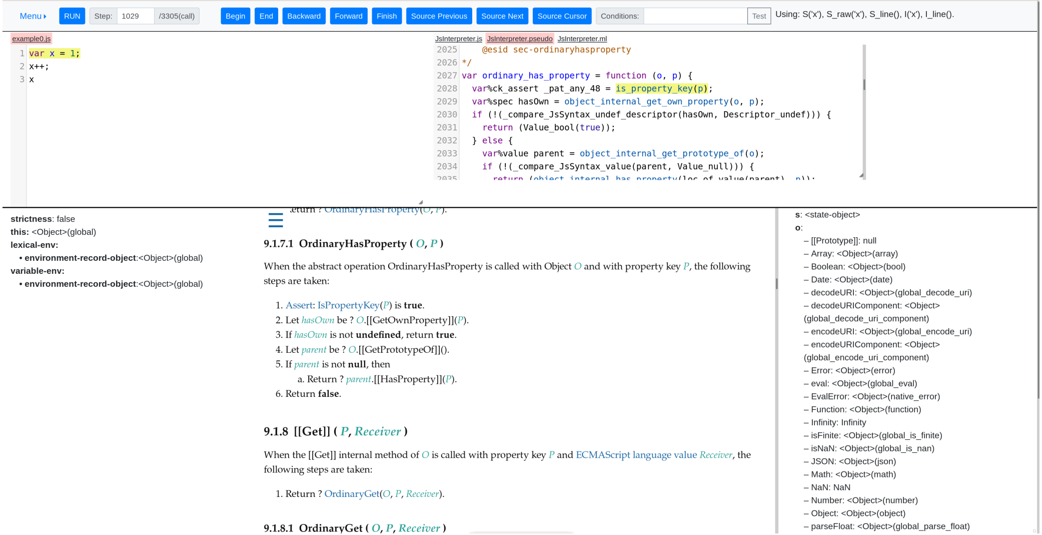
JSExplain

- instrument the generated JavaScript to record *events*
  - `Enter` (enter a function)
  - `CreateCtx(ctx)` (new function scope)
  - `Add(ident,value)` (`let` binding)
  - `Return` (return from a function)
- executing the instrumented interpreter generates a trace of events
- web tool to navigate these traces

Architecture


AST of interpreted program
Esprima
interpreted program
generator
web page
Libraries (JS)
Interpreter and libraries (OCaml)
Interpreter with traces (JS)
trace
tracing generator

Demo



```
var x = 1;
x++;
x
```

```
    @esid sec-ordinaryhasproperty
*/
var ordinary_has_property = function (o, p) {
  var%ck_assert _pat_any_48 = is_property_key(p);
  var%spec hasOwn = object_internal_get_own_property(o, p);
  if (!(_compare_JsSyntax_undef_descriptor(hasOwn, Descriptor_undef))) {
    return (Value_bool(true));
  } else {
    var%value parent = object_internal_get_prototype_of(o);
    if (!(_compare_JsSyntax_value(parent, Value_null))) {
      return (object_internal_has_property(loc_of_value(parent), p));
```

strictness: false
this: <Object>(global)
lexical-env:
- **environment-record-object**:<Object>(global)

variable-env:
- **environment-record-object**:<Object>(global)

eturn ? OrdinaryHasProperty(*O*, *P*).

9.1.7.1 OrdinaryHasProperty (*O*, *P*)

When the abstract operation OrdinaryHasProperty is called with Object *O* and with property key *P*, the following steps are taken:

1. Assert: IsPropertyKey(*P*) is **true**.
2. Let *hasOwn* be ? *O*.[[GetOwnProperty]](*P*).
3. If *hasOwn* is not **undefined**, return **true**.
4. Let *parent* be ? *O*.[[GetPrototypeOf]]().
5. If *parent* is not **null**, then
    a. Return ? *parent*.[[HasProperty]](*P*).
6. Return **false**.

9.1.8 [[Get]] (*P*, *Receiver*)

When the [[Get]] internal method of *O* is called with property key *P* and ECMAScript language value *Receiver*, the following steps are taken:

1. Return ? OrdinaryGet(*O*, *P*, *Receiver*).

9.1.8.1 OrdinaryGet (*O*, *P*, *Receiver*)

s: <state-object>
o:
– [[Prototype]]: null
– Array: <Object>(array)
– Boolean: <Object>(bool)
– Date: <Object>(date)
– decodeURI: <Object>(global_decode_uri)
– decodeURIComponent: <Object>(global_decode_uri_component)
– encodeURI: <Object>(global_encode_uri)
– encodeURIComponent: <Object>(global_encode_uri_component)
– Error: <Object>(error)
– eval: <Object>(global_eval)
– EvalError: <Object>(native_error)
– Function: <Object>(function)
– Infinity: Infinity
– isFinite: <Object>(global_is_finite)
– isNaN: <Object>(global_is_nan)
– JSON: <Object>(json)
– Math: <Object>(math)
– NaN: NaN
– Number: <Object>(number)
– Object: <Object>(object)
– parseFloat: <Object>(global_parse_float)

`http://jsexplain.gforge.inria.fr/index.html`

Skeletal Semantics

The Problem with JSCert

```
(** If statement (12.5) *)

| red_stat_if : forall S C e1 t2 t3opt y1 o,
    red_spec S C (spec_expr_get_value_conv spec_to_boolean e1) y1 ->
    red_stat S C (stat_if_1 y1 t2 t3opt) o ->
    red_stat S C (stat_if e1 t2 t3opt) o

| red_stat_if_1_true : forall S0 S C t2 t3opt o,
    red_stat S C t2 o ->
    red_stat S0 C (stat_if_1 (vret S true) t2 t3opt) o

| red_stat_if_1_false : forall S0 S C t2 t3 o,
    red_stat S C t3 o ->
    red_stat S0 C (stat_if_1 (vret S false) t2 (Some t3)) o

| red_stat_if_1_false_implicit : forall S0 S C t2,
    red_stat S0 C (stat_if_1 (vret S false) t2 None) (out_ter S resvalue_empty)
```

- 900 mutually inductive rules
- inversion during an induction runs out of memory

Ingredients of a Semantics

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{tt} \qquad \sigma, s1 \Downarrow o}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o} \qquad \frac{\sigma, e \Downarrow v \qquad v = \texttt{ff} \qquad \sigma, s2 \Downarrow o}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o}$$

Evaluate `if e then s1 else s2` in state σ

1. Let `v` the result of evaluating `e` in state σ.
2. If `v` is true, let `o` the result of evaluating `s1` in state σ.
3. If `v` is false, let `o` the result of evaluating `s2` in state σ.
4. Return `o`.

Ingredients of a Semantics

Sequence

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{tt} \qquad \sigma, s1 \Downarrow o}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o} \qquad\qquad \frac{\sigma, e \Downarrow v \qquad v = \texttt{ff} \qquad \sigma, s2 \Downarrow o}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o}$$

Evaluate `if e then s1 else s2` in state σ

1. Let `v` the result of evaluating `e` in state σ.
2. If `v` is true, let `o` the result of evaluating `s1` in state σ.
3. If `v` is false, let `o` the result of evaluating `s2` in state σ.
4. Return `o`.

Sequence

Ingredients of a Semantics

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{tt} \qquad \overset{\text{Recursion}}{\boxed{\sigma, s1 \Downarrow o}}}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o} \qquad \frac{\sigma, e \Downarrow v \qquad v = \texttt{ff} \qquad \sigma, s2 \Downarrow o}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o}$$

Evaluate `if e then s1 else s2` in state σ

1. Let `v` the result of evaluating `e` in state σ.
2. If `v` is true, let `o` the result of evaluating `s1` in state σ. Recursion
3. If `v` is false, let `o` the result of evaluating `s2` in state σ.
4. Return `o`.

Ingredients of a Semantics

Choice

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{tt} \qquad \sigma, s1 \Downarrow o}{\sigma, \texttt{if}\ e\ \texttt{then}\ s1\ \texttt{else}\ s2 \Downarrow o} \qquad \frac{\sigma, e \Downarrow v \qquad v = \texttt{ff} \qquad \sigma, s2 \Downarrow o}{\sigma, \texttt{if}\ e\ \texttt{then}\ s1\ \texttt{else}\ s2 \Downarrow o}$$

Evaluate `if e then s1 else s2` in state σ

1. Let `v` the result of evaluating `e` in state σ.
2. If `v` is true, let `o` the result of evaluating `s1` in state σ.
3. If `v` is false, let `o` the result of evaluating `s2` in state σ.
4. Return `o`.

Choice

Ingredients of a Semantics

Atom

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{tt} \qquad \sigma, s1 \Downarrow o}{\sigma, \texttt{if}\ e\ \texttt{then}\ s1\ \texttt{else}\ s2 \Downarrow o} \qquad \frac{\sigma, e \Downarrow v \qquad v = \texttt{ff} \qquad \sigma, s2 \Downarrow o}{\sigma, \texttt{if}\ e\ \texttt{then}\ s1\ \texttt{else}\ s2 \Downarrow o}$$

Evaluate `if e then s1 else s2` in state σ

1. Let `v` the result of evaluating `e` in state σ.
2. If `v` is true, let `o` the result of evaluating `s1` in state σ.
3. If `v` is false, let `o` the result of evaluating `s2` in state σ.
4. Return `o`.

Atom

Ingredients of a Semantics

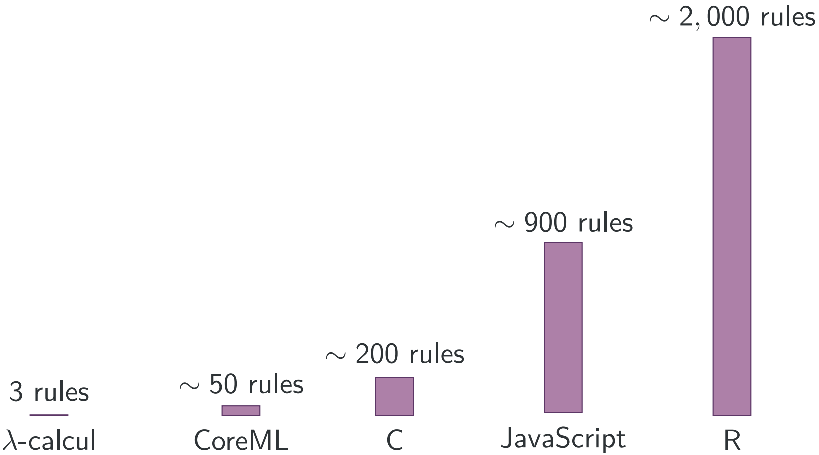
- Structure: sequence, recursion, choice
- Atoms

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{tt} \qquad \sigma, s1 \Downarrow o}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o} \qquad\qquad \frac{\sigma, e \Downarrow v \qquad v = \texttt{ff} \qquad \sigma, s2 \Downarrow o}{\sigma, \texttt{if}\ \ e\ \ \texttt{then}\ \ s1\ \ \texttt{else}\ \ s2 \Downarrow o}$$

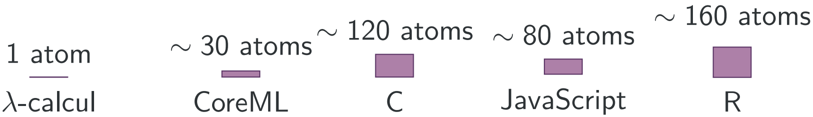
Evaluate `if e then s1 else s2` in state σ

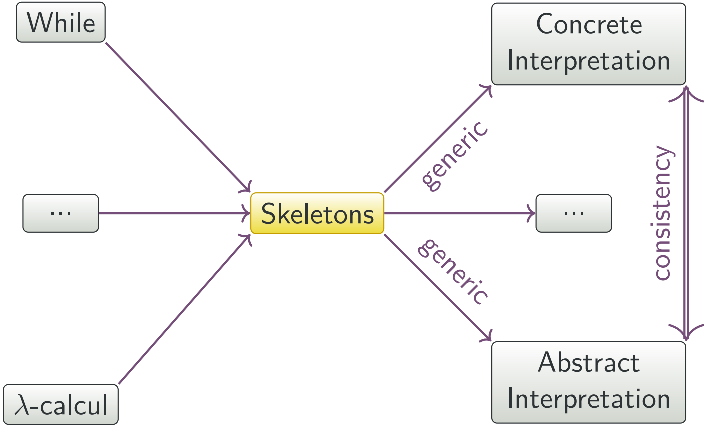
1. Let `v` the result of evaluating `e` in state σ.
2. If `v` is true, let `o` the result of evaluating `s1` in state σ.
3. If `v` is false, let `o` the result of evaluating `s2` in state σ.
4. Return `o`.

Size of Semantics (Number of Rules)



Size of Semantics (Number of Atoms)





While

⋯

λ-calcul

Skeletons

generic

generic

Concrete Interpretation

⋯

Abstract Interpretation

consistency

Skeletal Semantics

- Simple framework capturing the structure of semantics
- Generic definition of interpretations
- Proof techniques to relate interpretations
- Generation of an OCaml interpreter
- Generation of an analyzer
- Needs to be applied to JavaScript
- Technical talk this afternoon

What's Next

Beyond JSExplain and Skeletal Semantics

- Application to other languages (MLExplain)
- Generation of *-explain from Skeletal Semantics
- Analyses requested by TC39
  - Invariants
  - Object Capabilities

Conclusion

- Formalizing semantics can be fun (and fruitful)!
- JavaScript is an ideal candidate: complex and precise

Questions?

- https://tc39.github.io/
- http://www.jscert.org/
- https://gitlab.inria.fr/star-explain/
- http://skeletons.inria.fr/