

# JSCert

A Formalisation of JavaScript in Coq

Martin BODIN

CC7125-1 / MA7125-1

15<sup>th</sup> and 20<sup>th</sup> of November

# What we have seen in the course

- The Imp programming language.

## Semantics in the form of inductive predicates

`aeval`, `ceval`, etc.

## Semantics in the form of functions

- With fuel: `ceval_step` from `ImpCEvalFun.v`.
  - This last function is extractable to OCaml (see `Extraction.v`).
- With proof argument: `no_whiles_terminating` from `Imp.v`.

Let us apply this to other  
languages!

How difficult can it be?

# The world is complex

C, JavaScript, R, Python, Php, ...

- R

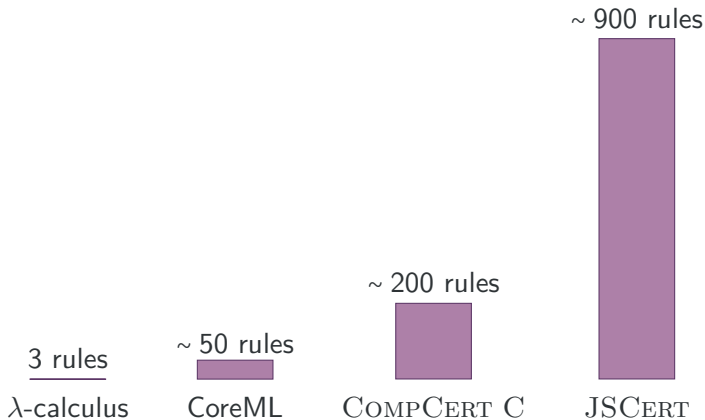
```
1  "(" <- function (a) a + 1
2  (1)      # Returns 2.
```

- Python

```
1  a = 256
2  b = 256
3  a is b      # Returns True
4  a = 257
5  b = 257
6  a is b      # Returns False
7  a = 257; b = 257
8  a is b      # Returns True
```

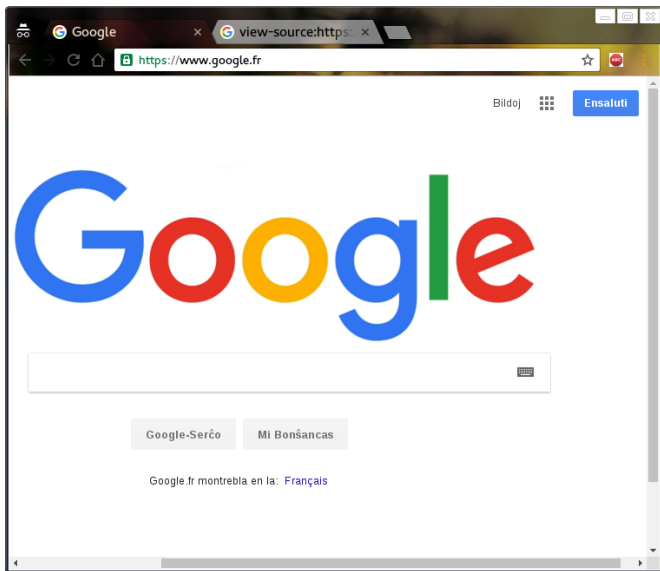
# Semantic Sizes

JavaScript is full of exceptions...

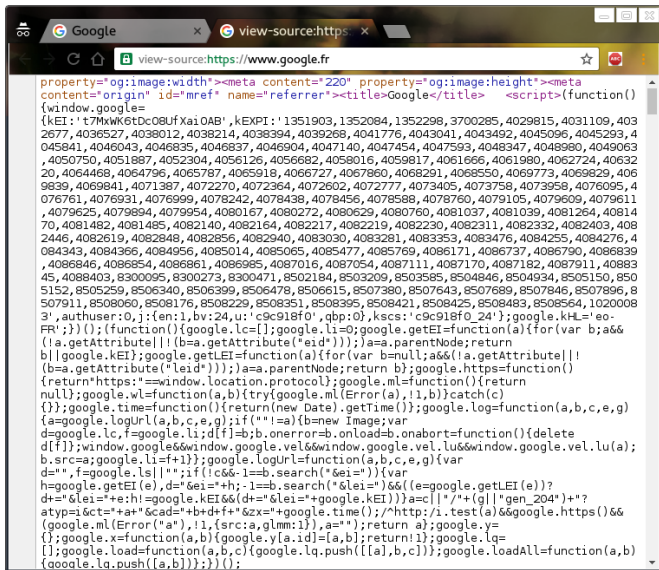


But why JavaScript?

# The Language of the Web



# The Language of the Web

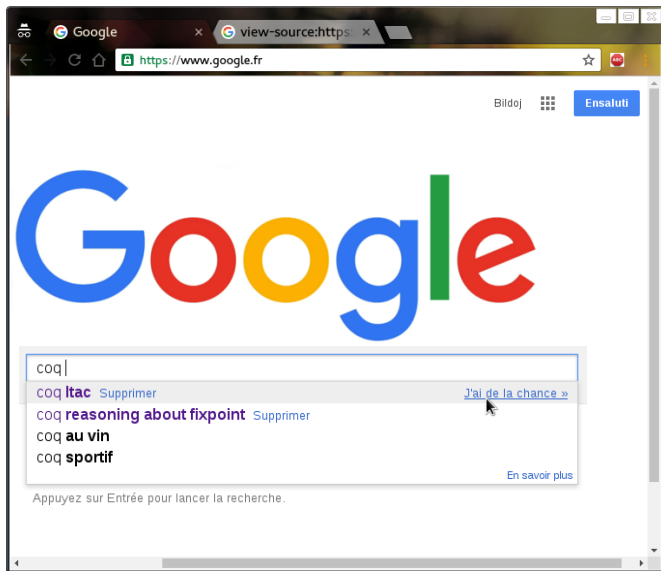


```
property="og:image:width"><meta content="220" property="og:image:height"><meta
content="origin" id="mref" name="referrer"><title>Google</title> <script>(function()
{window.google=
{kEI: 't7MxWK6tDc08UfXaiOAB', kEXPI: '1351903,1352084,1352298,3700285,4029815,4031109,403
2677,4036527,4038012,4038214,4038394,4039268,4041776,4043041,4043492,4045096,4045293,4
045841,4046043,4046835,4046837,4046904,4047140,4047454,4047593,4048347,4048980,4049063
,4050750,4051887,4052304,4056126,4056682,4058016,4059817,4061666,4061980,4062724,40632
20,4064468,4064796,4065787,4065918,4066727,4067860,4068291,4068550,4069773,4069829,406
9839,4069841,4071387,4072270,4072364,4072602,4072777,4073405,4073758,4073958,4076095,4
076761,4076931,4076999,4078242,4078438,4078456,4078588,4078760,4079105,4079609,4079611
,4079625,4079894,4079954,4080167,4080272,4080629,4080760,4081037,4081039,4081264,40814
70,4081482,4081485,4082140,4082164,4082217,4082219,4082230,4082311,4082332,4082403,408
2446,4082619,4082848,4082856,4082940,4083030,4083281,4083353,4083476,4084255,4084276,4
084343,4084366,4084956,4085014,4085065,4085477,4085769,4086171,4086737,4086790,4086839
,4086846,4086854,4086861,4086985,4087016,4087054,4087111,4087170,4087182,4087911,40883
45,4088403,8300095,8300273,8300471,8502184,8503209,8503585,8504846,8504934,8505150,850
5152,8505259,8506340,8506399,8506478,8506615,8507380,8507643,8507689,8507846,8507896,8
507911,8508060,8508176,8508229,8508351,8508395,8508421,8508425,8508483,8508564,1020008
3',authuser:0,j:{en:1,bv:24,u:'e9c918f0',qbp:0},kscs:'c9c918f0_24'};google.kHL='e9
FR'};})();(function(){google.lc=[];google.li=0;google.getEI=function(a){for(var b;a&&
(!a.getAttribute||!(b=a.getAttribute("eid")));a=a.parentNode;return
b|google.kEI;google.getLEI=function(a){for(var b=null;a&&(!a.getAttribute||
(b=a.getAttribute("leid")));a=a.parentNode;return b};google.https=function()
{return"https://"===window.location.protocol};google.ml=function(){return
null};google.wl=function(a,b){try{google.ml(Error(a),!1,b)}catch(c)
{}};google.time=function(){return(new Date).getTime();};google.log=function(a,b,c,e,g)
{a=google.logUrl(a,b,c,e,g);if(!a){b=new Image;var
d=google.lc,f=google.li;d[f]=b;b.onerror=b.onload=b.onabort=function(){delete
d[f]};window.google.lc&&window.google.vel&&window.google.vel.lu&&window.google.vel.lu(a);
b.src=a;google.li=f+1};google.logUrl=function(a,b,c,e,g){var
d="",f=google.lc;if(!c&&1=b.search("&ei="))){var
h=google.getEI(d,d+"&ei="+h;-1=b.search("&lei=")&&(e=google.getLEI(e)?
d+"&lei="+e:h!=google.kEI&&(d+"&lei="+google.kEI))a=c||"/+(g|"gen_204")+?"
atyp=i&ct="+a"&cad="+b+d+f+"&zx="+google.time();/"http://i.test(a)&&google.https()&
(google.ml(Error("a"),!1,{src:a,glmm:1}),a="");return a};google.y=
{};google.x=function(a,b){google.y[a.id]=[a,b];return 1};google.lq=
[];google.load=function(a,b,c){google.lq.push([a,b,c]);google.loadAll=function(a,b)
{google.lq.push([a,b]);}}})();
```

7,500 lines of JAVASCRIPT code!



# The Language of the Web



7,500 lines of JAVASCRIPT code!

# JAVASCRIPT and Mashups

Location de voiture dès **29€** / jour

Réserver maintenant et économisez

Map data ©2016 Google

Back to search results

Displaying 1 - 45 of 76 properties.

1 night (25 Nov - 26 Nov)

**Appartement Moderne**  
★★★★  
Very good 8.0  
Rates from **EUR 47**  
with taxes and fees  
[More details](#)

**Appartement Moderne**  
★★★★  
Free Wi-Fi  
Rates from **EUR 51**  
with taxes and fees  
[More details](#)

**Appartement Moderne**  
★★★★  
Excellent 8.3  
Rates from **EUR 60**  
with taxes and fees  
[More details](#)

1 - 45 [Next](#)



You +1'd this



# JAVASCRIPT and Mashups

Map

CLICK TO SEARCH THIS AREA

Location de voiture dès 29€ / jour

Réserver maintenant et économisez

Advertisement



You +1'd this



Tweet



Like

Social plugins

External search engine

Back to search results

Displaying 1 - 45 of 76 properties.

1 night (25 Nov - 26 Nov)

Royal Palace Hotel  
3 stars  
Very good 8.0  
Rates from EUR 47 with taxes and fees  
More details

Appartement Central Rennes Les Grands  
3 stars  
Free Wi-Fi  
Rates from EUR 51 with taxes and fees  
More details

Royal Palace Centre  
3 stars  
Excellent 9.3  
Rates from EUR 60 with taxes and fees  
More details

1 - 45

Next

# JAVASCRIPT is *Specified*



Note: JSCert is only about ECMAScript 5.1  
(<https://www.ecma-international.org/ecma-262/5.1>).

# Type Coercions in JavaScript

# The base types of JavaScript

- *Locations* ( $\simeq$  pointers);
- (Floating point) *numbers*: 42, 1.8e-35, -0, +0, +Infinity, -Infinity, NaN, etc.;
- (UTF-16 character) *strings*;
- *Booleans*: true and false;
- null (this is not a location);
- undefined (this is a defined value).

# The base types of JavaScript

- *Locations* ( $\simeq$  pointers);
  - (Floating point) *numbers*: 42, 1.8e-35, -0, +0, +Infinity, -Infinity, NaN, etc.;
  - (UTF-16 character) *strings*;
  - *Booleans*: true and false;
  - null (this is not a location);
  - undefined (this is a defined value).
- 
- Locations points to objects.
  - Objects are finite maps from strings (called *fields*) to values.
  - Functions, arrays, etc. are just special kinds of objects.

# The base types of JavaScript

- *Locations* ( $\simeq$  pointers);
- (Floating point) *numbers*: 42, 1.8e-35, -0, +0, +Infinity, -Infinity, NaN, etc.;
- (UTF-16 character) *strings*;
- *Booleans*: true and false;
- null (this is not a location);
- undefined (this is a defined value).

```
1 var o = {} ;
2 "f" in o ;      // Returns false.
3 o.f, o["f"] ;  // Returns undefined.
4 o.f = 42 ;
5 "f" in o ;      // Returns true.
6 o.f = undefined ;
7 "f" in o ;      // Returns true.
8 delete o.f ;
9 "f" in o ;      // Returns false.
```



# JavaScript likes implicit type coercions

```
1 ((+![]+(!![])[([][[[]+![])[+[]]+([![]]+[[]][[]])][+!![]+[+[]]])+([]+
2 ![])[!![]+!![]]+([+!![])[+[]]+([+!![])[!![]+!![]+!![]]+([+!!
3 ]) [+!![]]) + [])[!![]+!![]+!![]]+(!![]+[[]][([+![])[+[]]+([![])+[
4 ] [ [] ]) ] + [ + [ ] ] ) + ( [ + ! [ ] ) [ ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + [ ] ] + ( [ + ! [ ]
5 ) [ ! [ ] + ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + ! [ ] ] ) [ + ! [ ] + [ + [ ] ] + ( [ + [ ] [ [ ] ] ) [ +
6 ! [ ] ] + ( [ + ! [ ] ) [ ! [ ] + ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + [ ] ] + ( [ + ! [ ] ) [ + ! [ ] ] +
7 ( [ + [ ] [ [ ] ] ) [ + [ ] ] + ( [ [ ( [ + ! [ ] ) [ + [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! [ ] + [ + [ ] ] ] +
8 ( [ + ! [ ] ) [ ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + [ ] ] + ( [ + ! [ ] ) [ ! [ ] + ! [ ] + ! [ ] ] + ( [
9 ] + ! [ ] ) [ + ! [ ] ] + [ ] ) [ ! [ ] + ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + [ ] ] + ( ! [ ] + [ ] [ ( [
10 + ! [ ] ) [ + [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! [ ] + [ + [ ] ] ] + ( [ + ! [ ] ) [ ! [ ] + ! [ ] ] + ( [ +
11 ! [ ] ] + [ + [ ] ] + ( [ + ! [ ] ) [ ! [ ] + ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + ! [ ] ] ) [ + ! [ ] + [
12 + [ ] ] + ( [ + ! [ ] ) [ + ! [ ] ] ) [ + ! [ ] + [ + [ ] ] + ( ! [ ] + [ ] [ ( [ + ! [ ] ) [ + [ ] ] + ( [
13 ! [ ] ] + [ ] [ [ ] ] ) [ + ! [ ] + [ + [ ] ] + ( [ + ! [ ] ) [ ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + [ ] ] + ( [
14 ] + ! [ ] ) [ ! [ ] + ! [ ] + ! [ ] ] + ( [ + ! [ ] ) [ + ! [ ] ] ) [ + ! [ ] + [ + [ ] ] + ( [ + [ ] [
15 [ ] ] ) [ ! [ ] + ! [ ] ] + ( [ ! [ ] ] + [ ] [ [ ] ] ) [ + ! [ ] + [ + [ ] ] + ( [ + [ ] [ [ ] ] ) [ + ! [ ] ] )
```

# JavaScript likes implicit type coercions

```
1 ((+![]+(!![])[([][[[]+![])[+[]]+([![]]+[[]][[]])][+!![]+[+[]]]]+([!+  
2 ![])[!![]+!![]]+([!+!![])[+[]]+([!+!![])[!![]+!![]+!![]]+([!+!!  
3 ![])[+!![]]]+[])[!![]+!![]+!![]]+(![]+[[]][([!+![])[+[]]+([![]]+[  
4 ![]][[]])][+!![]+[+[]]]]+([!+![])[!![]+!![]]+([!+![])[+[]]+([!+!![]  
5 )][!![]+!![]+!![]]+([!+!![])[+!![]]))[+!![]+[+[]]]]+([!+[]][[]])][+  
6 !![]]+([!+![])[!![]+!![]+!![]]+([!+!![])[+[]]+([!+!![])[+!![]]]+  
7 ([!+[]][[]])][+[]]+([!+![])[+[]]+([![]]+[[]][[]])][+!![]+[+[]]]]+  
8 ([!+![])[!![]+!![]]+([!+!![])[+[]]+([!+!![])[!![]+!![]+!![]]+([  
9 !+!![])[+!![]]]+[])[!![]+!![]+!![]]+([!+!![])[+[]]+(![]+[[]][([  
10 +![])[+[]]+([![]]+[[]][[]])][+!![]+[+[]]]]+([!+![])[!![]+!![]]+([!+  
11 !![])[+[]]+([!+!![])[!![]+!![]+!![]]+([!+!![])[+!![]]))[+!![]]+[  
12 +[]]]]+([!+!![])[+!![]]))[+!![]+[+[]]]]+(![]+[[]][([!+![])[+[]]+([  
13 ![]]+[[]][[]])][+!![]+[+[]]]]+([!+![])[!![]+!![]]+([!+!![])[+[]]+([  
14 !+!![])[!![]+!![]+!![]]+([!+!![])[+!![]]))[+!![]+[+[]]]]+([!+[]][  
15 ![])[!![]+!![]]+([![]]+[[]][[]])][+!![]+[+[]]]]+([!+[]][[]])][+!![])
```

1 "Bodin"

# JavaScript likes implicit type coercions

```
1 ((+![]+(!![])[(][[([+![])[+[]]+(!![])+[[]]])[+!![]+[+[]]]+([]+  
2 ![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!!  
3 [])[+!![]])+([])[!![]+!![]+!![]]+(!![]+[[]([+![])[+[]]+(!![])+[  
4 ]]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![]  
5 )[!![]+!![]+!![]]+([]+!![])[+!![]])+(+!![]+[+[]])+([]+[[]])+(+  
6 !![])+([]+![])[!![]+!![]+!![]]+([]+!![])[+[]]+([]+!![])[+!![]]+  
7 ([]+[[]])+([]+![])[!![]+!![]]+(!![]+[[]])+(!![]+[[]])+(+!![]+[+[]])+(  
8 ([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]  
9 +!![])[+!![]]+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!!  
10 +!![])[+[]]+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!!  
11 !![])[+[]]+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!!  
12 +[])]+([]+!![])[+!![]])+(+!![]+[+[]])+(!![]+[[]])+(!![]+[[]])+(  
13 ![]+[[]])+(!![]+[+[]])+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!!  
14 +!![])[!![]+!![]+!![]]+(!![]+[[]])+(!![]+[[]])+(!![]+[[]])+(!!  
15 [])[!![]+!![]]+(!![]+[[]])+(!![]+[[]])+(!![]+[+[]])+(!![]+[[]])
```

```
1 ("B" + "o" + "d" + "i" + "n")
```

# JavaScript likes implicit type coercions

```
1 ((+![]+(!![])[([][[[]+![])[+[]]+([![]]+[[]][[]])][+!![]+[+[]]])+([]+
2 ![])[!![]+!![]]+([+![])[+[]]+([+![])[!![]+!![]+!![]]+([+!!
3 ]) [+!![]]) + [])[!![]+!![]+!![]]+(![]+[[]][([+![])[+[]]+([![])+[
4 ]][[]])][+!![]+[+[]]) + ([+![])[!![]+!![]]+([+![])[+[]]+([+![])[
5 ])[!![]+!![]+!![]]+([+![])[+!![]]) + [])[+!![]]) + [])[+[]][[]]) + [
6 !![]]+([+![])[!![]+!![]+!![]]+([+![])[+[]]+([+![])[+!![]])[+!![]]+
7 ([+[]][[]]) + [])[+[]]+([][([+![])[+[]]+([![])+[[]][[]])][+!![]+[+[]]) +
8 ([+![])[!![]+!![]]+([+![])[+[]]+([+![])[!![]+!![]+!![]]+([
9 ]+!![])[+!![]]) + [])[!![]+!![]+!![]]+([+![])[+[]]+(![]+[[]][([
10 +![])[+[]]+([![])+[[]][[]])][+!![]+[+[]]) + ([+![])[!![]+!![]]+([+
11 !![])[+[]]+([+![])[!![]+!![]+!![]]+([+![])[+!![]]) + [])[+!![]]+[
12 +[]]) + ([+![])[+!![]]) + [])[+!![]][+!![]][+!![]]+([![])+[[]][([+![])[+[]]+([
13 ![])+[[]][[]])][+!![]+[+[]]) + ([+![])[!![]+!![]]+([+![])[+[]]+([
14 ]+!![])[!![]+!![]+!![]]+([+![])[+!![]]) + [])[+!![]]+([+[]][
15 ])[!![]+!![]]+([![])+[[]][[]]) + [])[+!![]]+([+[]][[]]) + [])[+!![]]
```

```
1 ("function Boolean(){}" [9] + "function filter(){}" [6] +
2 "undefined" [2] + "undefined" [5] + "undefined" [1])
```

# JavaScript likes implicit type coercions

```
1 ((+![]+(![]))([[]([[]+![])]+[[]]+([![]]+[[]]))[+!![]+[+[]]]+([]+  
2 ![])[![]+!![]]+([+![]])+[[]]+([+![]])[![]+!![]+!![]]+([+!!  
3 [])+[![]])+[![]+!![]+!![]]+([![]]+[[]([[]+![])]+[[]]+([![]]+[  
4 ])))[+!![]+[+[]]]+([]+![])[![]+!![]]+([+![]])[+[]]+([+![]][  
5 )[![]+!![]+!![]]+([+![]])[+!![]])][+!![]+[+[]]]+([]+[[]]))[+  
6 !![]]+([+![]])[![]+!![]+!![]]+([+![]])[+[]]+([+![]])[+!![]]+  
7 ([+[]]))+[[]]+([[]([[]+![])]+[[]]+([![]]+[[]]))[+!![]+[+[]]]+  
8 ([+![]])[![]+!![]]+([+![]])[+[]]+([+![]])[![]+!![]+!![]]+([  
9 ]+!![])[+!![]]+[![]+!![]+!![]]+([+![]])[+[]]+([![]]+[[]([[]  
10 +![])]+[[]]+([![]]+[[]]))[+!![]+[+[]]]+([]+![])[![]+!![]]+([+  
11 !![])[+[]]+([+![]])[![]+!![]+!![]]+([+![]])[+!![]])][+!![]+[  
12 +[]]]+([]+![])[+!![]])][+!![]+[+[]]]+(!![]+[[]([[]+![])]+[[]]+([  
13 ![]]+[[]]))[+!![]+[+[]]]+([]+![])[![]+!![]]+([+![]])[+[]]+([  
14 ]+!![])[![]+!![]+!![]]+([+![]])[+!![]])][+!![]+[+[]]]+([]+[[]  
15 ])))[![]+!![]]+([![]]+[[]]))[+!![]+[+[]]]+([]+[[]]))[+!![]])
```

```
1 ((false.constructor + "")[9] + ([].filter + "")[6] +  
2 "undefined"[2] + "undefined"[5] + "undefined"[1])
```

# JSCert

JavaScript is too complex: we really need Coq.

Versions of Coq	8.4	8.4pl6	8.4.6	...	8.6	8.7
The current Software Foundations	✗	✗	✗		✓	✓
JSCert	✗	✓	✗		✗	✗

- You really need to have the right Coq version.
- opam enables to deal with several Coq versions at the same time.

## Different versions of OCaml and Coq...

```
1  $ opam switch
2  --      -- 4.02.1 Official 4.02.1 release
3  --      -- 4.02.2 Official 4.02.2 release
4  4.02.3 C 4.02.3 Official 4.02.3 release
5  --      -- 4.03.0 Official 4.03.0 release
6  --      -- 4.04.0 Official 4.04.0 release
7  --      -- 4.04.1 Official 4.04.1 release
8  --      -- 4.04.2 Official 4.04.2 release
9  4.05.0 I 4.05.0 Official 4.05.0 release
10 4.06.0 I 4.06.0 Official 4.06.0 release
11 system I system System compiler (4.02.3)
12
13 $ coqc --version
14 The Coq Proof Assistant, version 8.4pl6 (November 2017)
15 compiled on Nov 02 2017 13:49:22 with OCaml 4.02.3
```



Everything is explained in <https://github.com/jscert/jscert>.

```
1 $ sudo apt install opam
2 $ opam init
3 $ opam switch 4.02.3
4 $ eval 'opam config env'
5 $ git clone https://github.com/jscert/jscert
6 $ cd jscert
7 $ make init
8 $ make
```

This takes some time: it is best to do it before Friday...

# Formal Semantics of JavaScript

## The ECMAScript standard



ECMA International, ed. *ECMAScript Language Specification. Standard ECMA-262, Edition 5.1.* 2011.

## Formal Semantics Close to ECMAScript



Sergio Maffeis, John C. Mitchell, and Ankur Taly. “An Operational Semantics for JAVASCRIPT”. In: *APLAS*. 2008.

## Formal Semantics Executable



Arjun Guha, Claudiu Saftoiu, and Shriram Krishnamurthi. “The Essence of JAVASCRIPT”. In: *ECOOP*. 2010.

## JSert



Martin Bodin et al. “A Trusted Mechanised JAVASCRIPT Specification”. In: *POPL*. 2014.

# Formal Semantics of JavaScript

## The ECMAScript standard



ECMA International, ed. *ECMAScript Language Specification. Standard ECMA-262, Edition 5.1.* 2011.

## Formal Semantics Close to ECMAScript



Sergio Maffeis, John C. Mitchell, and Ankur Taly. “An Differences

- Languages: English, rules, program;
- Semantic styles: big-step, small-step, other;
- Ways to relate to JavaScript.
  - What is JavaScript: standard or implementation?

ii.

## JSert



Martin Bodin et al. “A Trusted Mechanised JAVASCRIPT Specification”. In: *POPL*. 2014.

# What is JavaScript: standard or implementation?

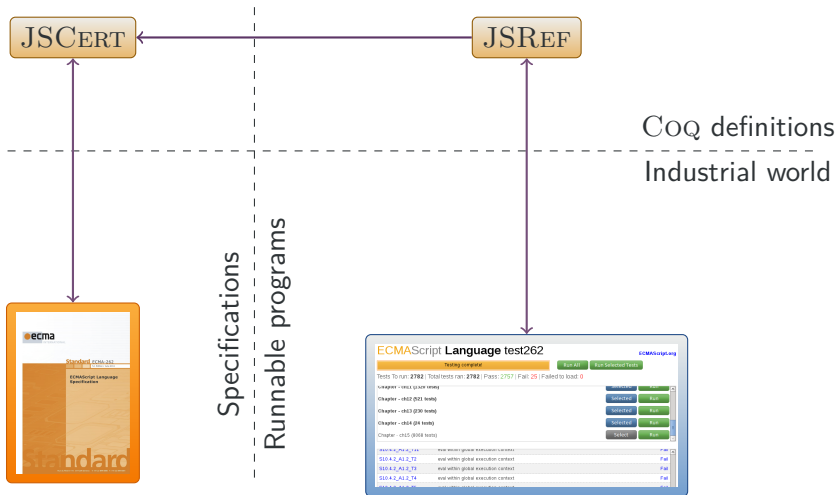
## 15.3.4.2 `Function.prototype.toString ( )`

An implementation-dependent representation of the function is returned. This representation has the syntax of a `FunctionDeclaration`. Note in particular that the use and placement of white space, line terminators, and semicolons within the representation `String` is implementation-dependent.

([www.ecma-international.org/ecma-262/5.1/#sec-15.3.4.2](http://www.ecma-international.org/ecma-262/5.1/#sec-15.3.4.2))

## In practice

```
1  eval (Function.prototype.toString.call (String))
2  // SyntaxError: Unexpected identifier
3  // Here is the returned string:
4  // 'function String() { [native code] }'
```





“`s1 ; s2`” is evaluated as follows.

- 1 Let  $o_1$  be the result of evaluating `s1`.
- 2 If  $o_1$  is an exception, return  $o_1$ .
- 3 Let  $o_2$  be the result of evaluating `s2`.
- 4 If an exception  $V$  was thrown, return (*Throw*,  $V$ , *empty*).
- 5 If  $o_2.value$  is empty, let  $V = o_1.value$ , otherwise let  $V = o_2.value$ .
- 6 Return ( $o_2.type$ ,  $V$ ,  $o_2.target$ ).

Big-step leads to a lot of repetitions.



“s1 ; s2” is evaluated as follows.

- 1 Let  $o_1$  be the result of evaluating s1.
- 2 If  $o_1$  is an exception, return  $o_1$ .
- 3 Let  $o_2$  be the result of evaluating s2.
- 4 If an exception  $V$  was thrown, return (*Throw*,  $V$ , empty).
- 5 If  $o_2.value$  is empty, let  $V = o_1.value$ , otherwise let  $V = o_2.value$ .
- 6 Return ( $o_2.type$ ,  $V$ ,  $o_2.target$ ).

Conditions

Evaluation of a subterm

Returning a result

# Pretty-Big-Step, Intuition

“ $s_1 ; s_2$ ” is evaluated as follows.

- 1 Let  $o_1$  be the result of evaluating  $s_1$ .
- 2 If  $o_1$  is an exception, return  $o_1$ .
- 3 Let  $o_2$  be the result of evaluating  $s_2$ .

Conditions

Evaluation of a subterm

Returning a result



# Pretty-Big-Step, Intuition

“s1 ; s2” is evaluated as follows.

- 1 Let  $o_1$  be the result of evaluating  $s_1$ .
- 2 If  $o_1$  is an exception, return  $o_1$ .
- 3 Let  $o_2$  be the result of evaluating  $s_2$ .

Conditions

Evaluation of a subterm

Returning a result

$$\frac{\text{SEQ-1}(s_1, s_2) \quad \sigma, s_1 \Downarrow o_1 \quad o_1, \text{seq}_1 \ s_2 \Downarrow o}{\sigma, \text{seq} \ s_1 \ s_2 \Downarrow o}$$

$$\frac{\text{SEQ-2}(s_2)}{o_1, \text{seq}_1 \ s_2 \Downarrow o_1} \quad \text{abort } o_1$$

$$\frac{\text{SEQ-3}(s_2) \quad o_1, s_2 \Downarrow o_2 \quad o_1, o_2, \text{seq}_2 \Downarrow o}{o_1, \text{seq}_1 \ s_2 \Downarrow o}$$

–abort  $o_1$       ...

## Definition

- We are in big-step style;
- Each rule has at most two inductive calls;
- If a rule conditionnaly applies, its condition can be decided using only its input.

## Consequences

- The outputs of inductive calls are never inspected;
- Partially evaluated terms have to be added, as in small-step. We call these partially evaluated terms “extended terms”.



“ $e1 + e2$ ” is evaluated as follows.

- 1 Let *lref* be the result of evaluating  $e1$ .
- 2 Let *lval* be the result of *GetValue*(*lref*).
- 3 Let *rref* be the result of evaluating  $e2$ .
- 4 Let *rval* be the result of *GetValue*(*rref*).
- 5 Let *lprim* be the result of *ToPrimitive*(*lref*).
- 6 Let *rprim* be the result of *ToPrimitive*(*rref*).
- 7 If *Type*(*lprim*) is *String* or *Type*(*rprim*) is *String*, then return the concatenation of *ToString*(*lprim*) and *ToString*(*rprim*).
- 8 Return the addition (with seven special cases) of *ToNumber*(*lprim*) and *ToNumber*(*rprim*).

# Exception handling is implicit in expressions



“ $e_1 + e_2$ ” is evaluated as follows.

- 1 Let *lref* be the result of evaluating  $e_1$ .
- 2 Let *lval* be the result of *GetValue*(*lref*).
- 3 Let *rref* be the result of evaluating  $e_2$ .
- 4 Let *rval* be the result of *GetValue*(*rref*).
- 5 Let *lprim* be the result of *ToPrimitive*(*lref*).
- 6 Let *rprim* be the result of *ToPrimitive*(*rref*).
- 7 If *Type*(*lprim*) is *String* or *Type*(*rprim*) is *String*, then return the concatenation of *ToString*(*lprim*) and *ToString*(*rprim*).
- 8 Return the addition (with seven special cases) of *ToNumber*(*lprim*) and *ToNumber*(*rprim*).

```
{toString: function(){ return true }} + 42)
```

# Exception handling is implicit in expressions



“e1 + e2” is evaluated as follows.

- 1 Let *lref* be the result of evaluating e1.
- 2 Let *lval* be the result of *GetValue*(*lref*).
- 3 Let *rref* be the result of evaluating e2.
- 4 Let *rval* be the result of *GetValue*(*rref*).
- 5 Let *lprim* be the result of *ToPrimitive*(*lref*).
- 6 Let *rprim* be the result of *ToPrimitive*(*rref*).
- 7 If *Type*(*lprim*) is *String* or *Type*(*rprim*) is *String*, then return the concatenation of *ToString*(*lprim*) and *ToString*(*rprim*).
- 8 Return the addition (with seven special cases) of *ToNumber*(*lprim*) and *ToNumber*(*rprim*).

```
{toString: function(){ return true }} + (42).toString()
```

```

1  Inductive red_expr
2    : state → execution_ctx → ext_expr → out → Prop :=
3
4  | red_expr_binary_op : forall S C op e1 e2 y1 o ,
5    regular_binary_op op →
6    red_spec S C (spec_expr_get_value e1) y1 →
7    red_expr S C (expr_binary_op_1 op y1 e2) o →
8    red_expr S C (expr_binary_op e1 op e2) o
9
10 | red_expr_binary_op_1 : forall S0 S C op v1 e2 y1 o ,
11   red_spec S C (spec_expr_get_value e2) y1 →
12   red_expr S C (expr_binary_op_2 op v1 y1) o →
13   red_expr S0 C (expr_binary_op_1 op (ret S v1) e2) o
14
15 | red_expr_binary_op_2 : forall S0 S C op v1 v2 o ,
16   red_expr S C (expr_binary_op_3 op v1 v2) o →
17   red_expr S0 C (expr_binary_op_2 op v1 (ret S v2)) o

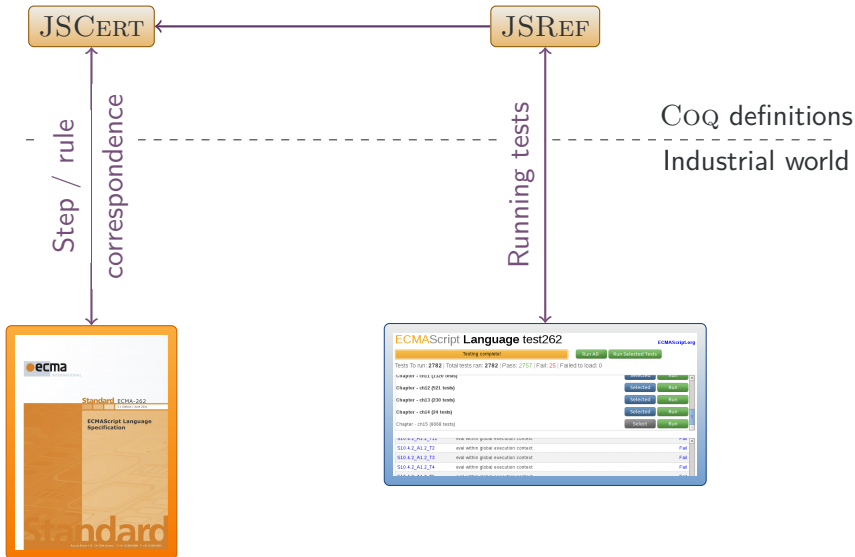
```

```
1 Inductive red_expr
2   : state → execution_ctx → ext_expr → out → Prop :=
3
4   | red_expr_abort : forall S C exte o,
5     out_of_ext_expr exte = Some o →
6     abort o →
7     ~ abort_intercepted_expr exte →
8     red_expr S C exte o
9
10  | red_expr_binary_op_2 : forall S0 S C op v1 v2 o,
11    red_expr S C (expr_binary_op_3 op v1 v2) o →
12    red_expr S0 C (expr_binary_op_2 op v1 (ret S v2)) o
```

```
1 | red_expr_binary_op_add : forall S C v1 v2 y1 o,  
2   red_spec S C (spec_convert_twice (spec_to_primitive_auto v1)  
3     (spec_to_primitive_auto v2)) y1 →  
4   red_expr S C (expr_binary_op_add_1 y1) o →  
5   red_expr S C (expr_binary_op_3 binary_op_add v1 v2) o  
6  
7 | red_expr_binary_op_add_1_string : forall S0 S C v1 v2 y1 o,  
8   (type_of v1 = type_string ∨ type_of v2 = type_string) →  
9   red_spec S C (spec_convert_twice (spec_to_string v1)  
10     (spec_to_string v2)) y1 →  
11   red_expr S C (expr_binary_op_add_string_1 y1) o →  
12   red_expr S0 C (expr_binary_op_add_1 (ret S (v1, v2))) o  
13  
14 | red_expr_binary_op_add_string_1 : forall S0 S C s1 s2 s,  
15   s = String.append s1 s2 →  
16   red_expr S0 C (expr_binary_op_add_string_1  
17     (ret S (value_prim s1, value_prim s2))) (out_ter S s)
```



```
1 | red_expr_binary_op_add : forall S C v1 v2 y1 o,  
2   red_spec S C (spec_convert_twice (spec_to_primitive_auto v1)  
3     (spec_to_primitive_auto v2)) y1 →  
4   red_expr S C (expr_binary_op_add_1 y1) o →  
5   red_expr S C (expr_binary_op_3 binary_op_add v1 v2) o  
6  
7 | red_expr_binary_op_add_1_number : forall S0 S C v1 v2 y1 o,  
8   ~ (type_of v1 = type_string \ / type_of v2 = type_string) →  
9   red_spec S C (spec_convert_twice (spec_to_number v1)  
10     (spec_to_number v2)) y1 →  
11   red_expr S C (expr_puremath_op_1 JsNumber.add y1) o →  
12   red_expr S0 C (expr_binary_op_add_1 (ret S (v1,v2))) o
```



```
1 Definition run_seq S (s1 s2 : stat) : result :=  
2   if_success (run_stat S s1) (fun S1 o1 =>  
3     if_success (run_stat S1 s2) (fun S2 o2 =>  
4       (* ... *))).
```

```
1 Definition run_seq S (s1 s2 : stat) : result :=  
2   if_success (run_stat S s1) (fun S1 o1 =>  
3     if_success (run_stat S1 s2) (fun S2 o2 =>  
4       (* ... *))).
```

```
1 Inductive out :=  
2   | out_ter : state → res → out.  
3  
4 Inductive result :=  
5   | result_some : out → result  
6   | result_not_yet_implemented : result  
7   | result_impossible : result  
8   | result_bottom : state → result.
```

```
1 Definition if_result_some W (K : out → result) : result :=
2   match W with
3   | result_some o => K o
4   | _ => W
5   end.
6
7 Definition if_ter W (K : state → res → result) : result :=
8   if_result_some W (fun o =>
9     match o with
10    | out_ter S0 R => K S0 R
11    | _ => result_some o
12    end).
13
14 Definition if_success W (K : state → resvalue → result) :=
15   if_ter W (fun S0 R =>
16     match res_type R with
17     | restype_normal => K S0 (res_value R)
18     | _ => res_out (out_ter S0 R)
19     end).
```

JSR<sub>EF</sub> is executable and can be tested.



```
1 while (1 === 1){
2     var v = "reached" ;
3     break
4 }
5 if (v !== "reached")
6     $ERROR ("v === 'reached'. Actual: v === " + v)
```

# JSR<sub>EF</sub> is executable and can be tested.



```
1 while (1 === 1){
2   var v = "reached" ;
3   break
4 }
5 if (v !== "reached")
6   $ERROR ("v === 'reached'. Actual: v === " + v)
```

```
1 function $ERROR (str) {
2   try {
3     __$ERROR__ = __$ERROR__ + " | " + str
4   } catch(ex) { __$ERROR__ = str }
5 }
```

- Easy to check the existence of the global variable `__$ERROR__`.

# JSREF is executable and can be tested.



```
1 while (1 === 1){  
2   var v = "reached" ;  
3   break  
4 }  
5 if (v !== "reached")  
6   $ERROR ("v === 'reached'. Actual: v === " + v)
```

## Possible Outputs

	Test should succeed	Test should fail
Test succeeded	✓	✗
Test failed	✗	✓
“Impossible” result	✗	✗
Out of fuel	Abort (no conclusion)	
Not yet implemented		



# Correctness Theorem

```
1 Theorem run_javascript_correct : forall p o,  
2   run_javascript p = Some o →  
3   red_javascript p o.
```



Martin Bodin and Alan Schmitt. “A Certified JavaScript Interpreter”. In: *JFLA*. 2013.

```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.
```



```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.
```



## Inductive hypotheses

```
1 Record runs_type_correct runs := {  
2   runs_type_correct_expr : forall S C e o,  
3     runs_type_expr runs S C e = o →  
4     red_expr S C e o;  
5   runs_type_correct_stat : forall S C t o,  
6     runs_type_stat runs S C t = o →  
7     red_stat S C t o;  
8   (* ... *) }.
```

# Automation is Mandatory

```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.  
5 Proof.  
6   introv HR. run red_seq_1.  
7     subst. applys* red_seq_2.  
8     subst. applys* red_seq_3. (* ... *)  
9 Qed.
```

# Automation is Mandatory

```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.
```

5 Proof.

```
6   introv HR. run red_seq_1.  
7     subst. applys* red_seq_2.  
8     subst. applys* red_seq_3. (* ... *)
```

9 Qed.

```
1 Ltac run rule :=  
2   let o1 := fresh "o1" in let R1 := fresh "R1" in  
3   run_pre o1 R1;  
4   (apply rule with o1 || apply rule with R1);  
5   try (run_post; run_inv; try assumption).
```

# Automation is Mandatory

```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.
```

5 Proof.

```
6   introv HR. run red_seq_1.  
7     subst. applys* red_seq_2.  
8     subst. applys* red_seq_3. (* ... *)
```

9 Qed.

```
1 Ltac run_pre o1 R1 :=  
2   match goal with H: ?T = result_some _ |- _ =>  
3     let h := match T with  
4       | runs_type_expr _ _ _ => constr:(runs_type_correct_expr)  
5       | if_success _ _ => constr:(if_success_out)  
6       (* ... *)  
7     end in  
8     (destruct (h H) as [o1 R1] || set (R1 := h H))  
9   end.
```

# Automation is Mandatory

```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.
```

5 Proof.

```
6   introv HR. run red_seq_1.  
7     subst. applys* red_seq_2.  
8     subst. applys* red_seq_3. (* ... *)
```

9 Qed.

```
1 Definition if_success_post (K : _ → _ → result) o o1 :=  
2   eqabort o1 o \/  
3     exists S rv, o1 = out_ter S (res_normal rv) /\ K S rv = o.
```

```
4 Definition isout W (Predi : out → Prop) :=  
5   exists o1, W = res_out o1 /\ Pred o1.
```

```
6  
7 Lemma if_success_out : forall W K o,  
8   if_success W K = res_out o →  
9   isout W (if_success_post K o).
```

# Automation is Mandatory

```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.
```

5 Proof.

```
6   introv HR. run red_seq_1.  
7     subst. applys* red_seq_2.  
8     subst. applys* red_seq_3. (* ... *)
```

9 Qed.

```
1 Ltac run_post :=  
2   match goal with  
3   | H: if_success_post _ _ _ |- _ => destruct H  
4   (* ... *)  
5   end.
```



# Automation is Mandatory

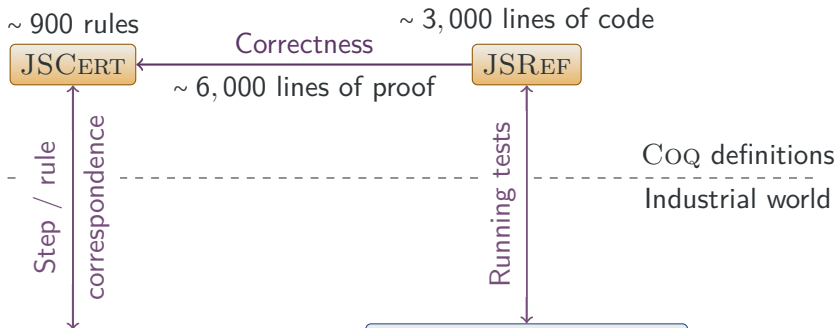
```
1 Lemma run_seq_correct : forall runs S s1 s2 o,  
2   runs_type_correct runs →  
3   run_seq runs S s1 s2 = o →  
4   red_stat S (seq s1 s2) o.
```

5 Proof.

```
6   introv HR. run red_seq_1.  
7     subst. applys* red_seq_2.  
8     subst. applys* red_seq_3. (* ... *)
```

9 Qed.

```
1 Ltac run_inv :=  
2   match goal with  
3   | H: out_div = out_ter __ |- __ => inversion H  
4   | H: out_ter __ = out_ter __ |- __ => inversion H  
5   (* ... *)  
6   end.
```



~ 900 steps  
~ 200 pages



5,126 tests passed

- 1 This Course
- 2 But why JavaScript?
- 3 Type Coercions in JavaScript
- 4 Compiling JSCert
- 5 JSCert