

JSCert

A Formalisation of JavaScript in Coq

Martin BODIN

CC7125-1 / MA7125-1

15th and 20th of November

- JavaScript is complex;
- JavaScript is specified by ECMAScript;
- Translating ECMAScript into big-step is long and not scalable;
- We can translate each steps of ECMAScript into one pretty-big-step rule;
- JSCert is a translation of the core of JavaScript into Coq/pretty-big-step;
- JSCert is accompanied with an interpreter, JSRef;
- We can run JSRef against test suites.

Why is JavaScript so Complex?

- Initially, JavaScript was designed for small scripts done by non-professional programmers.
- Also, only designed in 10 days.
- *Don't break the web!*
- There are actually efforts to make JavaScript simpler:
 - `for (/* ... */ of /* ... */) iteratively replacing`
`for (/* ... */ in /* ... */);`
 - the strict mode;
 - etc.
- Inertia is the biggest enemy here, but we can fight it progressively.

Why Big-step wouldn't work in JSCert?

What is great about JavaScript

To do anything about JavaScript, you have to be able to *scale*.

- JavaScript forces us to do things in a scalable way.
- Big-step does not scale on ECMAScript.
- Pretty-big-step does.

Why Big-step wouldn't work in JSCert?

What is great about JavaScript

To do anything about JavaScript, you have to be able to *scale*.

- JavaScript forces us to do things in a scalable way.
- Big-step does not scale on ECMAScript.
- Pretty-big-step does.

Scaling

- In semantic size (900 rules just for the core, but what about libraries?);
- In program size (7,500 lines just for Google's main page?);
- In time (ECMAScript 6, 7, 8, ES.Next, ...).

Making Coq Proofs Scale

Why proof automation works?

```
i 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  
  
i 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  
  
i 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)  
  
i 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  
  
i 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  
  
i 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  
  
i 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  
  
i 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)  
  
i 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  
  
i red_stat_while_6_abort : forall S C labs e1 t2 rv R,  
  res_type R <> restype_normal ->  
  red_stat S C (stat_while_5 labs e1 t2 rv R) (out_ter S R)  
  
i 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  
  
i red_stat_abort : forall S C extt o,  
  out_of_ext_stat extt = Some o ->  
  abort o ->  
  ~ abort_intercepted_stat extt ->  
  red_stat 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).
```

JS CERT Specification Coverage

Chapter	Section	Page
1	Introduction	1
2	Scope	1
3	Conformance	1
4	Notation	1
5	ECMAScript	1
6	Language Overview	1
7	The Basic Types of ECMAScript	1
8	Forms and Identifiers	1
9	Method Conventions	1
10	Source Text	1
11	Lexical Conventions	1
12	Code Evaluation	1
13	Errors	1
14	Runtime Conventions and Identifiers	1
15	Standard Library	1
16	Core JavaScript	1
17	Annex A	1
18	Annex B	1

- Chapters 1–7: how to read ECMAScript;
- Chapters 8–14, 16: core JavaScript;
- Chapters 15: standard library.

The `for` (`/* ... */ in /* ... */`) construct

“is: for (lhse in e) s” is evaluated as follows.

- 1 Let *exprRef* be the result of evaluating *e*.
- 2 Let *exprValue* be *GetValue(exprRef)*.
- 3 If *exprValue* is `null` or `undefined`, return (*normal*, *empty*, *empty*).
- 4 Let *obj* be *ToObject(exprValue)*.
- 5 Let *V* = *empty*.
- 6 Repeat
 - 1 Let *P* be the name of the next property of *obj* whose *Enumerable* attribute is `true`. If there is no such property, return (*normal*, *V*, *empty*).
 - 2 Let *lhsRef* be the result of evaluating the lhse (it may be evaluated repeatedly).
 - 3 Call *PutValue(lhsRef, P)*.
 - 4 Let *stmt* be the result of evaluating *s*.
 - 5 If *stmt.value* is not empty, let *V* = *stmt.value*.
 - 6 If *stmt.type* is `break` and *stmt.target* is in the current label set,

The `for` (`/* ... */ in /* ... */`) construct

“is: for (lhs in e) s” is evaluated as follows.

⑥ Repeat

- ① Let P be the name of the next property of obj whose *Enumerable* attribute is `true`. If there is no such property, return $(normal, V, empty)$.
- ② Let $lhsRef$ be the result of evaluating the lhs (it may be evaluated repeatedly).
- ③ Call $PutValue(lhsRef, P)$.
- ④ Let $stmt$ be the result of evaluating s .
- ⑤ If $stmt.value$ is not empty, let $V = stmt.value$.
- ⑥ If $stmt.type$ is `break` and $stmt.target$ is in the current label set, return $(normal, V, empty)$.
- ⑦ If $stmt.type$ is not `continue` or $stmt.target$ is not in the current label set, then
 - ① If $stmt$ is an abrupt completion, return $stmt$.

The `for` (`/* ... */ in /* ... */`) construct

“is: for (lhse in e) s” is evaluated as follows.

⑥ Repeat

- ① Let P be the name of the next property of obj whose *Enumerable* attribute is `true`. If there is no such property, return (*normal*, V , *empty*).

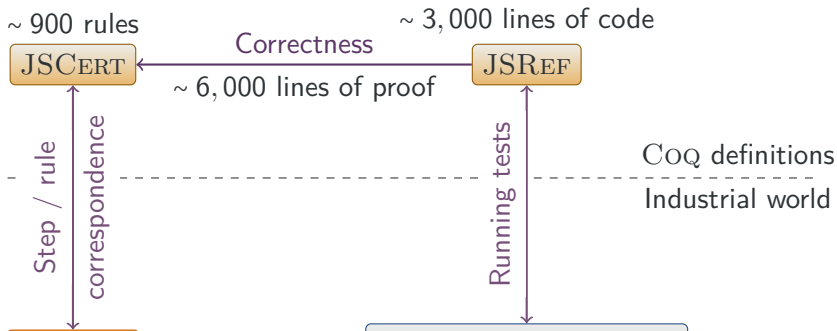
The mechanics and order of enumerating the properties (Step 1) is not specified. Properties of the object being enumerated may be deleted during enumeration, [they will then] not be visited. If new properties are added to the object being enumerated during enumeration, [they] are not guaranteed to be visited in the active enumeration. A property name must not be visited more than once in any enumeration. Enumerating the properties of an object includes enumerating properties of its prototype.

Zooming Out

What is the most difficult part of a formalisation?

You have to convince *other people* (often non-Coq people) that your semantics is the right one.

- This is actually not about Coq: Coq is useful for the people who will use your semantics, not for you;
- Be sure to understand the original language:
 - What is the most important: the specification or interpreters?
The language community?



~ 900 steps
~ 200 pages



5,126 tests passed
out of 11,725

Bugs found

Bugs in interpreters

- Invalid return values of `try { /* ... */ } finally { /* ... */ }` blocks;
- Changing dead code altered the final result.

Bugs in ECMAScript

- Broken algorithm;
- Some cases forgotten in the Enumerate method.

Bugs in test suites

- Tests checking the value of unspecified fields;
- Bugs in tests, mimicking implementation bugs.

Reporting bugs are great way to make people trust you!

<http://jscert.org/pop114/?full#20>

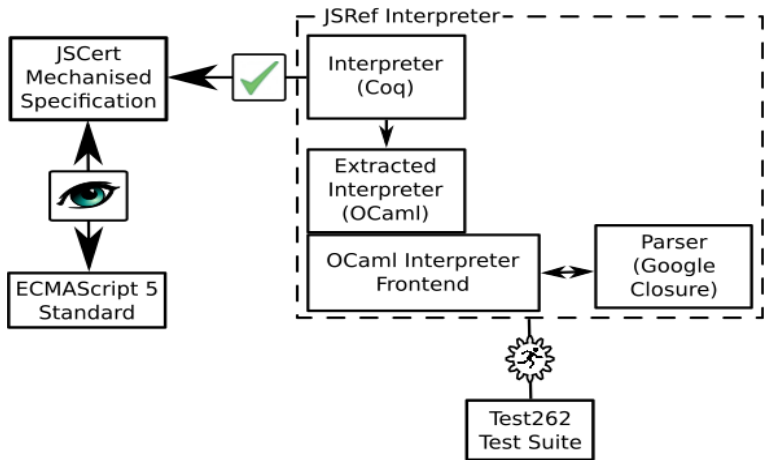
Increasing the Coverage of JSCert

Coverage of JSCert

ecma	ecma	ecma	ecma	ecma	ecma
Contents					
Introduction					
1 Scope					
2 Conformance					
3 Normative references					
4 Symbols					
5 The global object					
6 ECMAScript					
7 Basic and primitive types					
8 Reference operations					
9 Control flow operations					
10 The global object					
11 The global object					
12 The global object					
13 The global object					
14 The global object					
15 The global object					
16 The global object					
17 The global object					
18 The global object					
19 The global object					
20 The global object					
21 The global object					
22 The global object					
23 The global object					
24 The global object					
25 The global object					
26 The global object					
27 The global object					
28 The global object					
29 The global object					
30 The global object					
31 The global object					
32 The global object					
33 The global object					
34 The global object					
35 The global object					
36 The global object					
37 The global object					
38 The global object					
39 The global object					
40 The global object					
41 The global object					
42 The global object					
43 The global object					
44 The global object					
45 The global object					
46 The global object					
47 The global object					
48 The global object					
49 The global object					
50 The global object					
51 The global object					
52 The global object					
53 The global object					
54 The global object					
55 The global object					
56 The global object					
57 The global object					
58 The global object					
59 The global object					
60 The global object					
61 The global object					
62 The global object					
63 The global object					
64 The global object					
65 The global object					
66 The global object					
67 The global object					
68 The global object					
69 The global object					
70 The global object					
71 The global object					
72 The global object					
73 The global object					
74 The global object					
75 The global object					
76 The global object					
77 The global object					
78 The global object					
79 The global object					
80 The global object					
81 The global object					
82 The global object					
83 The global object					
84 The global object					
85 The global object					
86 The global object					
87 The global object					
88 The global object					
89 The global object					
90 The global object					
91 The global object					
92 The global object					
93 The global object					
94 The global object					
95 The global object					
96 The global object					
97 The global object					
98 The global object					
99 The global object					
100 The global object					

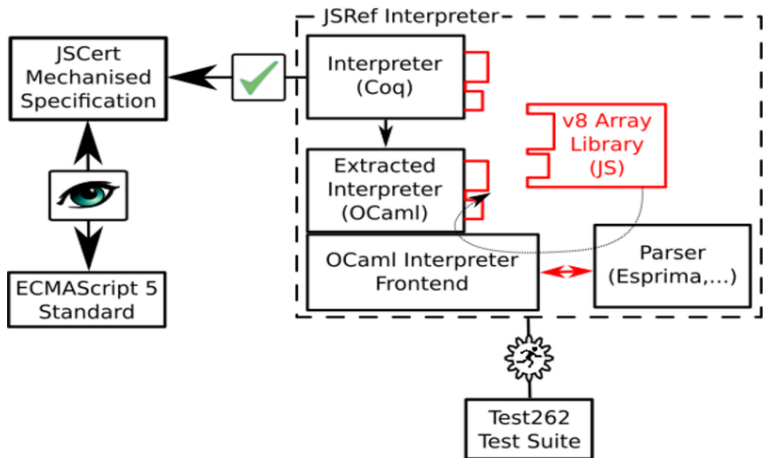
How to easily add Chapter 15?

Reusing already existing libraries



Philippa Gardner et al. "A Trusted Mechanised Specification of JAVASCRIPT: One Year On". In: *CAV*. 2015.

Reusing already existing libraries



Philippa Gardner et al. "A Trusted Mechanised Specification of JAVASCRIPT: One Year On". In: *CAV*. 2015.

Bisect

Let us test ECMAScript!

A code coverage tool for OCaml applied on JSRef, which is closed to ECMAScript:

```
1 let run_call_full max_step s c l vthis args =
2   (*[ 3311 ]*)(fun f0 fS n -> (*[ 3311 ]*)if n=0 then
3     (*[ 0 ]*)f0 () else (*[ 3311 ]*)fS (n-1))
4   (fun _ -> (*[ 0 ]*)Coq_result_bottom)
5   (fun max_step' ->
6     (*[ 3311 ]*)let run_expr' = run_expr max_step' in
7     (*[ 3311 ]*)if_some (run_object_method object_call_ s l)
8     (fun c0 ->
9       match c0 with
10      | Coq_call_default ->
11        (*[ 2555 ]*)entering_func_code runs' s c l vthis args
12      | Coq_call_after_bind -> (*[ 0 ]*)Coq_result_stuck
13      | Coq_call_prealloc b -> (*[ 756 ]*)run_call' s c b args))
14   max_step
```

Bisect

Let us test ECMAScript!

A code coverage tool for OCaml applied on JSRef, which is closed to ECMAScript:

```
1 let run_call_full max_step s c l vthis args =
2   (*[ 3311 ]*)(fun f0 fS n -> (*[ 3311 ]*)if n=0 then
3     (*[ 0 ]*)f0 () else (*[ 3311 ]*)fS (n-1))
4   (fun _ -> (*[ 0 ]*)Coq_result_bottom)
5   (fun max_step' ->
6     (*[ 3311 ]*)let run_expr' = run_expr max_step' in
7     (*[ 3311 ]*)if_some (run_object_method object_call_ s l)
8     (fun c0 ->
9       match c0 with
10      | Coq_call_default ->
11        (*[ 2555 ]*)entering_func_code runs' s c l vthis args
12      | Coq_call_after_bind -> (*[ 0 ]*)Coq_result_stuck
```

13 JSCert/JSRef enabled to test the coverage of the test suite for the
14 first time!



Using JSCert/JSRef



- <http://ajacs.inria.fr/jsexplain/driver.html>
- <https://github.com/jscert/jsexplain>

Central idea

- Everyone can read and understand JSRef;
- We could use the interpreter to explain JavaScript's behaviours;
- JSRef should be able to generate everything else.

Implement real browser behaviours

- The initial heap is different;
- Some behaviours (like `Function.prototype.toString ()`) are implementation-dependent.

We could formalise all these

- By adding special rules into JSCert;
- By adding a special argument to the predicates `red_expr` to denote the browser;
- By executing their JavaScript code to build another initial heap.

Complex, but possible



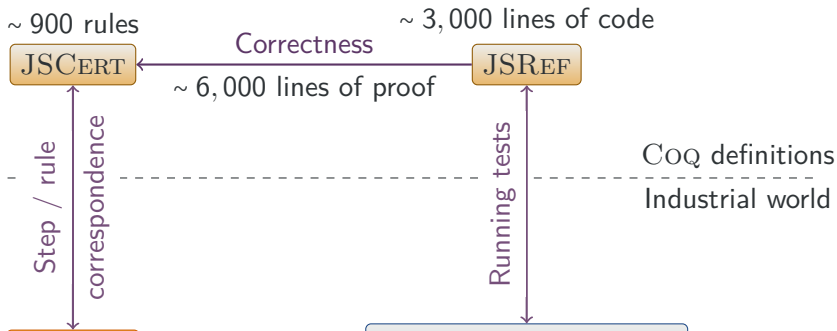
Philippa Gardner, Sergio Maffeis, and Gareth Smith. “Towards a Program Logic for JavaScript”. In: *POPL*. 2012.



Simon Holm Jensen, Anders Møller, and Peter Thiemann. “Type Analysis for JAVASCRIPT”. In: *SAS*. 2009.



Arlen Cox, Bor-Yuh Evan Chang, and Xavier Rival. “Automatic Analysis of Open Objects in Dynamic Language Programs”. In: *SAS*. 2014.



~ 900 steps
~ 200 pages



5,126 tests passed

- 1 Previously
- 2 Zooming Out
- 3 Increasing the Coverage of JSCert