JSCert

A Formalisation of JavaScript in Coq

Martin Bodin

CC7125-1 / MA7125-1

15^{th} and 20^{th} 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

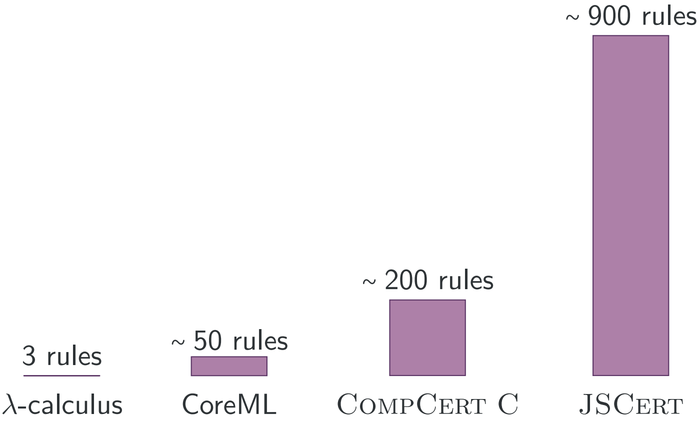
```r
"(" <- function (a) a + 1
(1)          # Returns 2.
```

- Python

```python
a = 256
b = 256
a is b      # Returns True
a = 257
b = 257
a is b      # Returns False
a = 257; b = 257
a is b      # Returns True
```

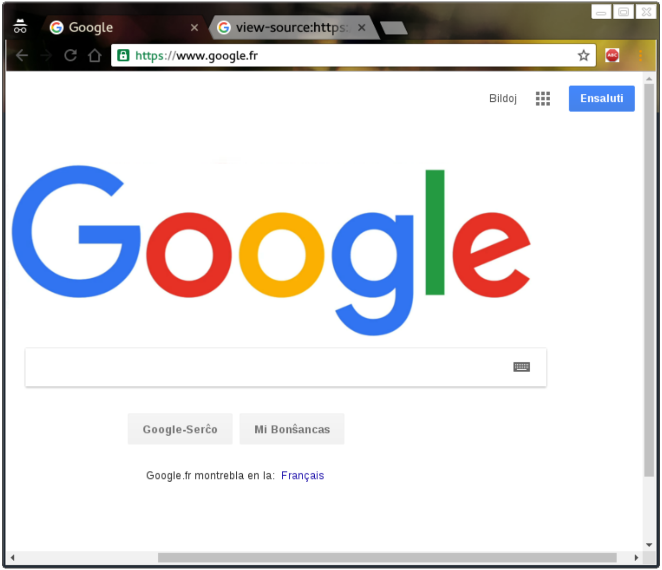
Semantic Sizes

JavaScript is full of exceptions...

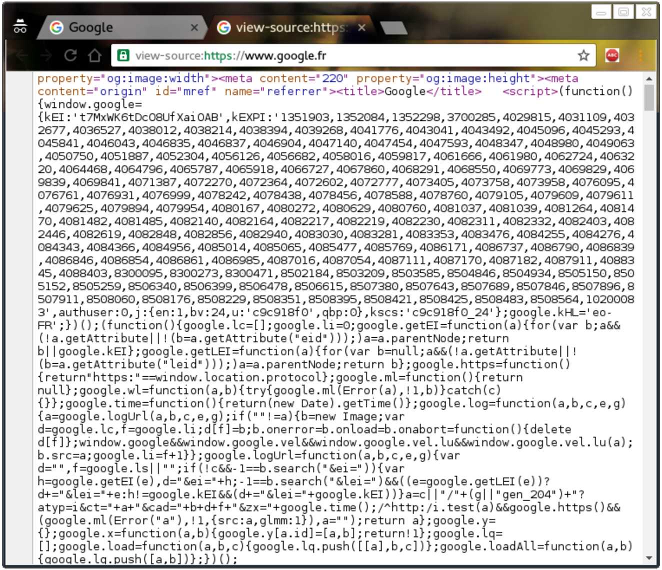


But why JavaScript?

The Language of the Web

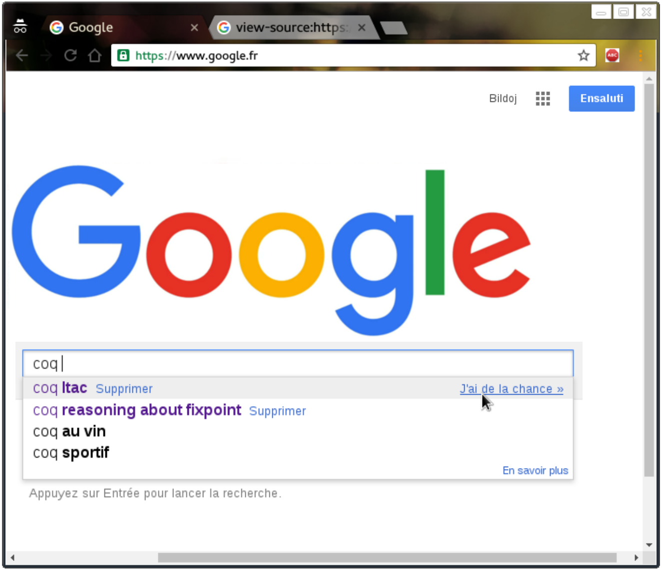


The Language of the Web



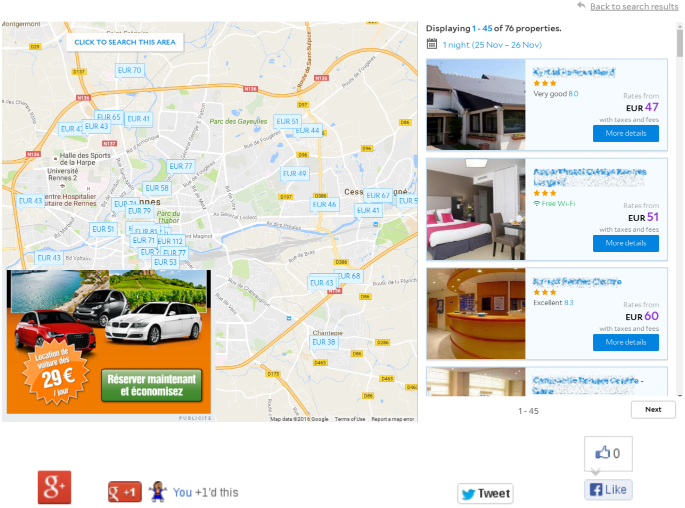
7,500 lines of JavaScript code!

The Language of the Web

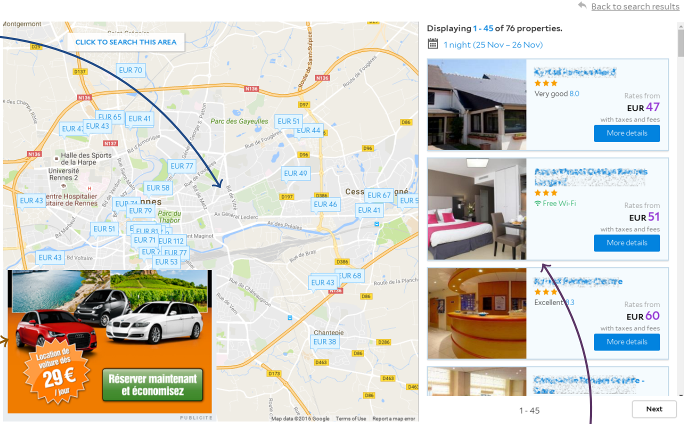


7,500 lines of JAVASCRIPT code!

JavaScript and Mashups



Map



Advertisement

Social plugins

External search engine

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* (≃ 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).

```
var o = {} ;
"f" in o ;      // Returns false.
o.f, o["f"] ;   // Returns undefined.
o.f = 42 ;
"f" in o ;      // Returns true.
o.f = undefined ;
"f" in o ;      // Returns true.
delete o.f ;
"f" in o ;      // Returns false.
```

JavaScript likes implicit type coercions

```
((+![]+(![])[([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+
![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!!
[])[+!![]]]+[])[!![]+!![]+!![]]+(!![]+[][([]+![])[+[]]+([![]]+[
][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![]
)[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][[]])[+
!![]]+([]+![])[!![]+!![]+!![]]+([]+!![])[+[]]+([]+!![])[+!![]]+
([]+[][[]])[+[]]+([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+
([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([
]+!![])[+!![]]]+[])[!![]+!![]+!![]]+([]+!![])[+[]]+(!![]+[][([]
+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+
!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[
+[]]]+([]+!![])[+!![]]])[+!![]+[+[]]]+(!![]+[][([]+![])[+[]]+([
![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([
]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][
[]])[!![]+!![]]+([![]]+[][[]])[+!![]+[+[]]]+([]+[][[]])[+!![]])
```

JavaScript likes implicit type coercions

```
((+![]+(![])[([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+
![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!!
[])[+!![]]]+[])[!![]+!![]+!![]]+(!![]+[][([]+![])[+[]]+([![]]+[
][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![]
)[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][[]])[+
!![]]+([]+![])[!![]+!![]+!![]]+([]+!![])[+[]]+([]+!![])[+!![]]+
([]+[][[]])[+[]]+([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+
([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([
]+!![])[+!![]]]+[])[!![]+!![]+!![]]+([]+!![])[+[]]+(!![]+[][([]
+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+
!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[
+[]]]+([]+!![])[+!![]]])[+!![]+[+[]]]+(!![]+[][([]+![])[+[]]+([
![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([
]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][
[]])[!![]+!![]]+([![]]+[][[]])[+!![]+[+[]]]+([]+[][[]])[+!![]])
```

```
”Bodin”
```

JavaScript likes implicit type coercions

```
((+![]+(![])[([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+
![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!!
[])[+!![]]]+[])[!![]+!![]+!![]]+(!![]+[][([]+![])[+[]]+([![]]+[
][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![]
)[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][[]])[+
!![]]+([]+![])[!![]+!![]+!![]]+([]+!![])[+[]]+([]+!![])[+!![]]+
([]+[][[]])[+[]]+([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+
([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([
]+!![])[+!![]]]+[])[!![]+!![]+!![]]+([]+!![])[+[]]+(!![]+[][([]
+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+
!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[
+[]]]+([]+!![])[+!![]]])[+!![]+[+[]]]+(!![]+[][([]+![])[+[]]+([
![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([
]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][
[]])[!![]+!![]]+([![]]+[][[]])[+!![]+[+[]]]+([]+[][[]])[+!![]])
```

```
(”B” + ”o” + ”d” + ”i” + ”n”)
```

JavaScript likes implicit type coercions

```
((+![]+(![])[([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+
![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!!
[])[+!![]]]+[])[!![]+!![]+!![]]+(!![]+[][([]+![])[+[]]+([![]]+[
][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![]
)[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][[]])[+
!![]]+([]+![])[!![]+!![]+!![]]+([]+!![])[+[]]+([]+!![])[+!![]]+
([]+[][[]])[+[]]+([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+
([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([
]+!![])[+!![]]]+[])[!![]+!![]+!![]]+([]+!![])[+[]]+(!![]+[][([]
+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+
!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[
+[]]]+([]+!![])[+!![]]])[+!![]+[+[]]]+(!![]+[][([]+![])[+[]]+([
![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([
]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][
[]])[!![]+!![]]+([![]]+[][[]])[+!![]+[+[]]]+([]+[][[]])[+!![]])
```

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

JavaScript likes implicit type coercions

```
((+![]+(![])[([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+
![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!!
[])[+!![]]]+[])[!![]+!![]+!![]]+(!![]+[][([]+![])[+[]]+([![]]+[
][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![]
)[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][[]])[+
!![]]+([]+![])[!![]+!![]+!![]]+([]+!![])[+[]]+([]+!![])[+!![]]+
([]+[][[]])[+[]]+([][([]+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+
([]+![])[!![]+!![]]+([]+!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([
]+!![])[+!![]]]+[])[!![]+!![]+!![]]+([]+!![])[+[]]+(!![]+[][([]
+![])[+[]]+([![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+
!![])[+[]]+([]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[
+[]]]+([]+!![])[+!![]]])[+!![]+[+[]]]+(!![]+[][([]+![])[+[]]+([
![]]+[][[]])[+!![]+[+[]]]+([]+![])[!![]+!![]]+([]+!![])[+[]]+([
]+!![])[!![]+!![]+!![]]+([]+!![])[+!![]]])[+!![]+[+[]]]+([]+[][
[]])[!![]+!![]]+([![]]+[][[]])[+!![]+[+[]]]+([]+[][[]])[+!![]])
```

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

JSCert

JavaScript is too complex: we really need Coq.

Before presenting JSCert

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...

```
$ opam switch
--      -- 4.02.1  Official 4.02.1 release
--      -- 4.02.2  Official 4.02.2 release
4.02.3   C 4.02.3  Official 4.02.3 release
--      -- 4.03.0  Official 4.03.0 release
--      -- 4.04.0  Official 4.04.0 release
--      -- 4.04.1  Official 4.04.1 release
--      -- 4.04.2  Official 4.04.2 release
4.05.0   I 4.05.0  Official 4.05.0 release
4.06.0   I 4.06.0  Official 4.06.0 release
system   I system  System compiler (4.02.3)

$ coqc --version
The Coq Proof Assistant, version 8.4pl6 (November 2017)
compiled on Nov 02 2017 13:49:22 with OCaml 4.02.3
```

Installing JSCert

Everything is explained in `https://github.com/jscert/jscert`.

```
$ sudo apt install opam
$ opam init
$ opam switch 4.02.3
$ eval ‘opam config env‘
$ git clone https://github.com/jscert/jscert
$ cd jscert
$ make init
$ 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.

JSCert

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?

ıi.

JSCert

Martin Bodin et al. "A Trusted Mechanised JavaScript Specification". In: *POPL*. 2014.

What is JavaScript: standard or implementation?

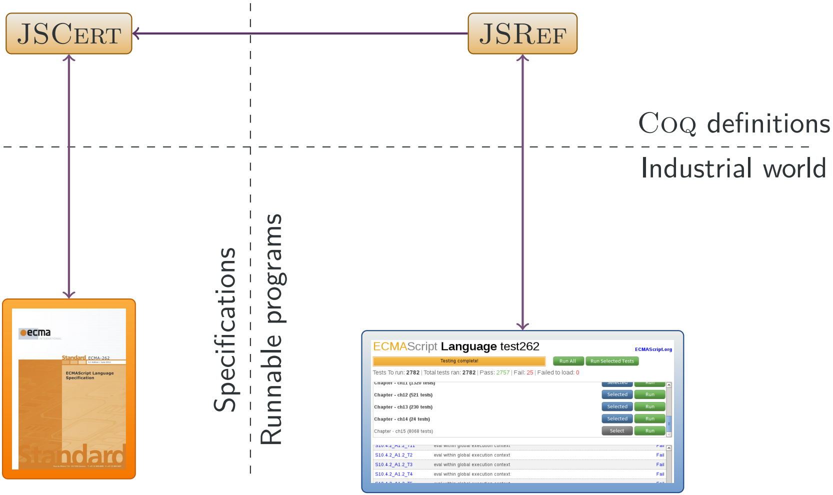
15.3.4.2 `Function.prototype.toString ( )`

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`)

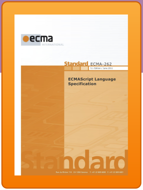
In practice

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

The JSCert Project



ECMAScript

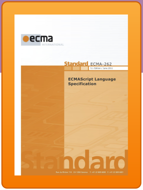


“`s1 ; s2`” is evaluated as follows.

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 $(\mathit{Throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

Big-step leads to a lot of repetitions.

ECMAScript



“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 $(\mathit{Throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

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 `s1`.
2. If o_1 is an exception, return o_1.
3. Let o_2 be the result of evaluating `s2`.

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 s1.
2. If o_1 is an exception, return o_1.
3. Let o_2 be the result of evaluating s2.

Conditions — Evaluation of a subterm — Returning a result

$$\text{SEQ-1}(s_1, s_2)\quad \frac{\sigma, s_1 \Downarrow o_1 \qquad o_1, seq_1\ \ s_2 \Downarrow o}{\sigma, seq\ \ s_1\ \ s_2 \Downarrow o}$$

$$\text{SEQ-2}(s_2)\quad \frac{}{o_1, seq_1\ \ s_2 \Downarrow o_1}\quad \textbf{abort}\ o_1$$

$$\text{SEQ-3}(s_2)\quad \frac{o_1, s_2 \Downarrow o_2 \qquad o_1, o_2, seq_2 \Downarrow o}{o_1, seq_1\ \ s_2 \Downarrow o}\quad \neg\textbf{abort}\ o_1$$

...

Pretty-Big-Step

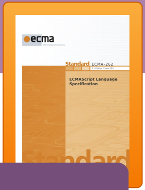
Definition

- 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".

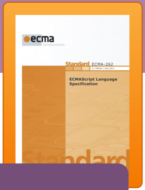
Exception handling is implicit in expressions



“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*).

Exception handling is implicit in expressions

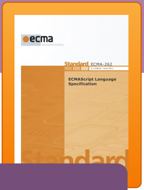


“`e1 + e2`” is evaluated as follows.

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)
```

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())
```

JSCert's Rules

```
Inductive red_expr
  : state → execution_ctx → ext_expr → out → Prop :=

  | red_expr_binary_op : forall S C op e1 e2 y1 o ,
    regular_binary_op op →
    red_spec S C (spec_expr_get_value e1) y1 →
    red_expr S C (expr_binary_op_1 op y1 e2) o →
    red_expr S C (expr_binary_op e1 op e2) o

  | red_expr_binary_op_1 : forall S0 S C op v1 e2 y1 o,
    red_spec S C (spec_expr_get_value e2) y1 →
    red_expr S C (expr_binary_op_2 op v1 y1) o →
    red_expr S0 C (expr_binary_op_1 op (ret S v1) e2) o

  | red_expr_binary_op_2 : forall S0 S C op v1 v2 o,
    red_expr S C (expr_binary_op_3 op v1 v2) o →
    red_expr S0 C (expr_binary_op_2 op v1 (ret S v2)) o
```

JSCert's Rules

```
Inductive red_expr
  : state → execution_ctx → ext_expr → out → Prop :=

  | red_expr_abort : forall S C exte o,
      out_of_ext_expr exte = Some o →
      abort o →
       ~ abort_intercepted_expr exte →
      red_expr S C exte o

  | red_expr_binary_op_2 : forall S0 S C op v1 v2 o,
    red_expr S C (expr_binary_op_3 op v1 v2) o →
    red_expr S0 C (expr_binary_op_2 op v1 (ret S v2)) o
```

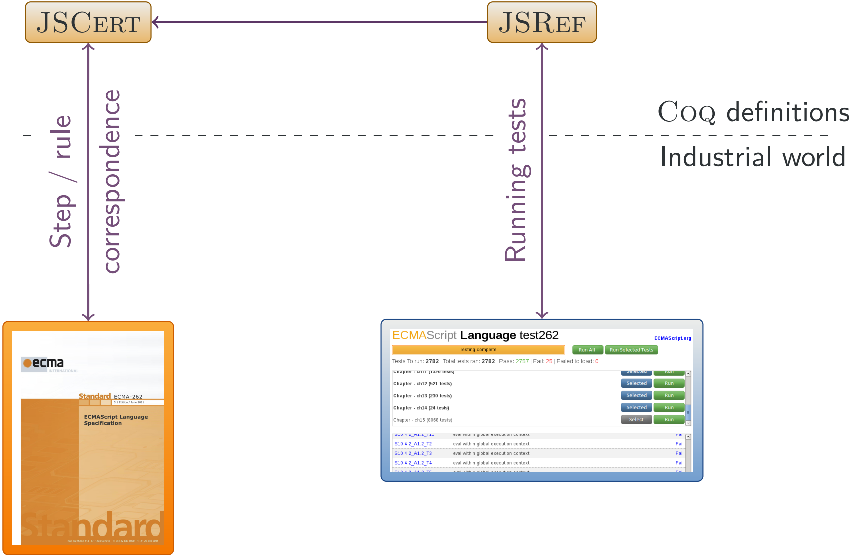
JSCert's Rules

```
| red_expr_binary_op_add : forall S C v1 v2 y1 o,
  red_spec S C (spec_convert_twice (spec_to_primitive_auto v1)
                (spec_to_primitive_auto v2)) y1 →
  red_expr S C (expr_binary_op_add_1 y1) o →
  red_expr S C (expr_binary_op_3 binary_op_add v1 v2) o

| red_expr_binary_op_add_1_string : forall S0 S C v1 v2 y1 o,
  (type_of v1 = type_string \/ type_of v2 = type_string) →
  red_spec S C (spec_convert_twice (spec_to_string v1)
                                   (spec_to_string v2)) y1 →
  red_expr S C (expr_binary_op_add_string_1 y1) o →
  red_expr S0 C (expr_binary_op_add_1 (ret S (v1, v2))) o

| red_expr_binary_op_add_string_1 : forall S0 S C s1 s2 s,
  s = String.append s1 s2 →
  red_expr S0 C (expr_binary_op_add_string_1
      (ret S (value_prim s1, value_prim s2))) (out_ter S s)
```

JSCert's Rules

```
| red_expr_binary_op_add : forall S C v1 v2 y1 o,
  red_spec S C (spec_convert_twice (spec_to_primitive_auto v1)
                (spec_to_primitive_auto v2)) y1 →
  red_expr S C (expr_binary_op_add_1 y1) o →
  red_expr S C (expr_binary_op_3 binary_op_add v1 v2) o

| red_expr_binary_op_add_1_number : forall S0 S C v1 v2 y1 o,
  ~ (type_of v1 = type_string \/ type_of v2 = type_string) →
  red_spec S C (spec_convert_twice (spec_to_number v1)
                                   (spec_to_number v2)) y1 →
  red_expr S C (expr_puremath_op_1 JsNumber.add y1) o →
  red_expr S0 C (expr_binary_op_add_1 (ret S (v1,v2))) o
```

The JSCert Project



Sequence in JSRef

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

Sequence in JSRef

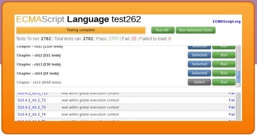
```
Definition run_seq S (s1 s2 : stat) : result :=
  if_success (run_stat S s1) (fun S1 o1 =>
    if_success (run_stat S1 s2) (fun S2 o2 =>
      (* ... *))).
```

```
Inductive out :=
  | out_ter : state → res → out.

Inductive result :=
  | result_some : out → result
  | result_not_yet_implemented : result
  | result_impossible : result
  | result_bottom : state → result.
```

Monads in JSRef

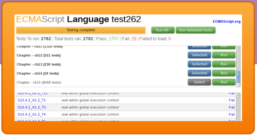
```
Definition if_result_some W (K : out → result) : result :=
  match W with
  | result_some o => K o
  | _ => W
  end.

Definition if_ter W (K : state → res → result) : result :=
  if_result_some W (fun o =>
    match o with
    | out_ter S0 R => K S0 R
    | _ => result_some o
    end).

Definition if_success W (K : state → resvalue → result) :=
  if_ter W (fun S0 R =>
    match res_type R with
    | restype_normal => K S0 (res_value R)
    | _ => res_out (out_ter S0 R)
    end).
```

JSREF is executable and can be tested.




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

JSRef is executable and can be tested.

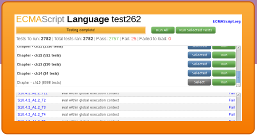



```
while (1 === 1){
  var v = "reached" ;
  break
}
if (v !== "reached")
  $ERROR ("v === 'reached'. Actual: v === " + v)
```

```
function $ERROR (str) {
    try {
        __$ERROR__ = __$ERROR__ + " | " + str
    } catch(ex) { __$ERROR__ = str }
}
```

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

JSREF is executable and can be tested.




```
while (1 === 1){
  var v = "reached" ;
  break
}
if (v !== "reached")
  $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

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

$$\begin{tikzcd} \text{JSCert} & & \text{JSRef} \arrow[ll, "\text{Correctness}"', "\supseteq"] \end{tikzcd}$$

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

Proof of Correctness

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

Proof of Correctness

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

Inductive hypotheses

```
Record runs_type_correct runs := {
  runs_type_correct_expr : forall S C e o,
    runs_type_expr runs S C e = o →
    red_expr S C e o;
  runs_type_correct_stat : forall S C t o,
    runs_type_stat runs S C t = o →
    red_stat S C t o;
  (* ... *) }.
```

Automation is Mandatory

```
Lemma run_seq_correct : forall runs S s1 s2 o,
  runs_type_correct runs →
  run_seq runs S s1 s2 = o →
  red_stat S (seq s1 s2) o.
Proof.
  introv HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

Automation is Mandatory

```
Lemma run_seq_correct : forall runs S s1 s2 o,
  runs_type_correct runs →
  run_seq runs S s1 s2 = o →
  red_stat S (seq s1 s2) o.
Proof.
  introv HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

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

Automation is Mandatory

```
Lemma run_seq_correct : forall runs S s1 s2 o,
  runs_type_correct runs →
  run_seq runs S s1 s2 = o →
  red_stat S (seq s1 s2) o.
Proof.
  introv HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

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

Automation is Mandatory

```
Lemma run_seq_correct : forall runs S s1 s2 o,
  runs_type_correct runs →
  run_seq runs S s1 s2 = o →
  red_stat S (seq s1 s2) o.
Proof.
  introv HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

```
Definition if_success_post (K : _ → _ → result) o o1 :=
  eqabort o1 o \/
    exists S rv, o1 = out_ter S (res_normal rv) /\ K S rv = o.
Definition isout W (Predi : out → Prop) :=
  exists o1, W = res_out o1 /\ Pred o1.

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

Automation is Mandatory

```
Lemma run_seq_correct : forall runs S s1 s2 o,
  runs_type_correct runs →
  run_seq runs S s1 s2 = o →
  red_stat S (seq s1 s2) o.
Proof.
  introv HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

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

Automation is Mandatory

```
Lemma run_seq_correct : forall runs S s1 s2 o,
  runs_type_correct runs →
  run_seq runs S s1 s2 = o →
  red_stat S (seq s1 s2) o.
Proof.
  introv HR. run red_seq_1.
    subst. applys* red_seq_2.
    subst. applys* red_seq_3. (* ... *)
Qed.
```

```
Ltac run_inv :=
  match goal with
  | H: out_div = out_ter _ _ |- _ => inversion H
  | H: out_ter _ _ = out_ter _ _ |- _ => inversion H
  (* ... *)
  end.
```

The JSCERT Project

~ 900 rules

~ 3,000 lines of code

JSCERT ← Correctness — JSREF

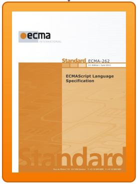
~ 6,000 lines of proof

Step / rule correspondence

Running tests

CoQ definitions

Industrial world



~ 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