

From JSCert to JSExplain and Beyond

Alan Schmitt, with Arthur Charguéraud (Inria Nancy) and Thomas Wood (Imperial College)

March 19, 2018

From JSCert

What is JSCert?

Two JavaScript semantics in Coq

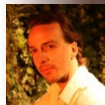
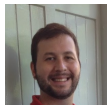
descriptive given a program and a result, say if they are related

executable given a program, compute the result

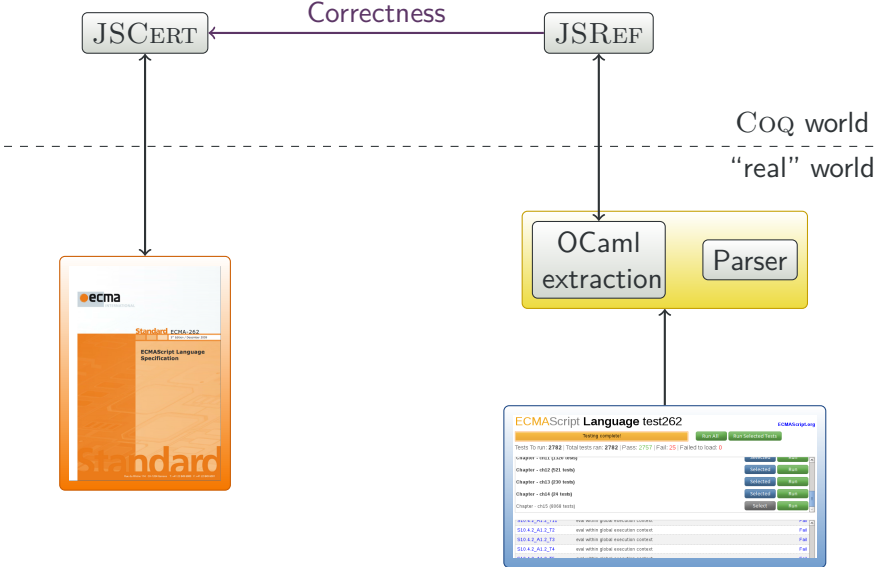
Correctness

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

- 2 years, 8 people
- 18 klocs of Coq



Overview



Positive Outcomes

- 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

```
'002632 (** val run_stat_while :  
'002633   int -> runs_type -> resvalue -> state -> execution_ctx -> label_set ->  
'002634   expr -> stat -> result **)  
'002635  
'002636 let rec run_stat_while max_step runs0 rv s c ls el t2 =  
'002637   (*[77]*) (fun f0 fS n -> (*[77]*) if n=0 then (*[0]*) f0 () else (*[77]*) fS (n-1))  
E002638   (fun ->  
'002639     (*[0]*) Coq_result_bottom)  
E002640   (fun max_step' ->  
'002641     (*[77]*) let run_stat_while' = run_stat_while max_step' runs0 in  
'002642     (*[77]*) if success_value runs0 c (runs0.runs_type_expr s c el) (fun s1 v1 ->  
'002643       (*[75]*) if convert_value_to_boolean v1  
'002644         then (*[59]*) if ter (runs0.runs_type_stat s1 c t2) (fun s2 r2 ->  
'002645           (*[59]*) let rvR = r2.res_value in  
'002646             (*[59]*) let rv' =  
'002647               if resvalue_comparable rvR Coq_resvalue_empty then (*[5]*) rv else (*[54]*)  
'002648               in  
'002649               (*[59]*) if normal_continue_or_break (Coq_result_out (Coq_out_ter (s2,  
'002650                 r2))) (fun r -> (*[41]*) res_label_in r ls) (fun s3 r3 ->  
'002651                 (*[40]*) run_stat_while' rv' s3 c ls el t2) (fun s3 r3 ->  
'002652                   (*[14]*) Coq_result_out (Coq_out_ter (s3, (res_ref rv'))))  
'002653                 else (*[16]*) Coq_result_out (Coq_out_ter (s1, (res_ref rv))))  
'002654     max_step
```

- Hard to keep pace with the standardisation
 - need to update two formalizations and a correctness proof
- JSCert inductive definition is too big
 - no inversion possible, preventing most proofs

- many low hanging fruits from an implementation close to the spec
- maintain a single artefact, derive other formats from it
- the Coq formalization should be usable for proofs

To 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

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`.

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

```
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 (Coq_value_prim w1)) Coq_type_string)
     || (type_compare (type_of (Coq_value_prim w2)) Coq_type_string)
  then
    let%string (s3, str1) = to_string s2 c (Coq_value_prim w1) in
    let%string (s4, str2) = to_string s3 c (Coq_value_prim w2) in
    res_out (Coq_out_ter (s4, (res_val (Coq_value_prim (Coq_prim_string (strappend str1 str2))))))
  else
    let%number (s3, n1) = to_number s2 c (Coq_value_prim w1) in
    let%number (s4, n2) = to_number s3 c (Coq_value_prim w2) in
    res_out (Coq_out_ter (s4, (res_val (Coq_value_prim (Coq_prim_number (n1 +. n2))))))
```

Compiled to JavaScript

- motivations: 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

```
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

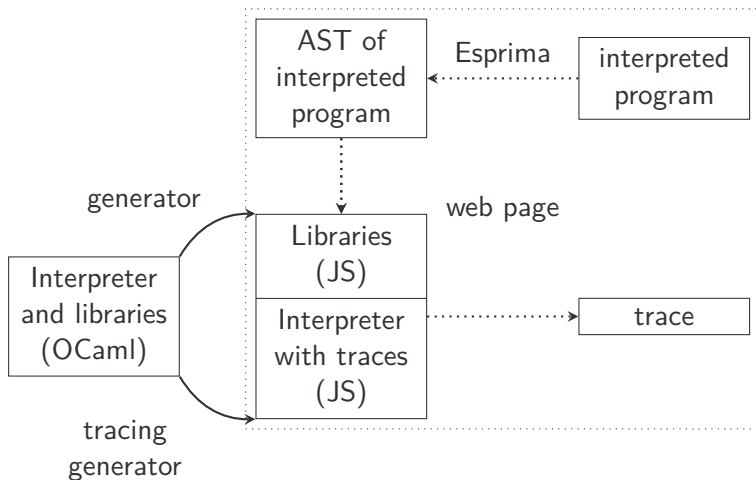
- to be readable while staying close to JavaScript
 - hide state and context
 - monadic extension of var
 - pattern matching
 - hide type changes

```
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);
  }
};
```

- 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



Demo

Program Selection

Load example: `var x = 1; x++; x` Load file: Choose File No file chosen

example0.js

```
1 var x = 1;
2 x++;
3 x
```

Current Program

Program Heap State

RUN Step: 0 / 2268 (enter)

Begin End Backward Forward Prev Next Finish Source Prev Source Next Source Cursor

Condition: `S_line() == 3 && S(') == 1` Reach Test Using: `S('x'), S_raw('x'), S_line(), I('x'), I_line()`.

js/interpreter.js js/interpreter.pseudo js/interpreter.m

```
4819
4820 };
4821
4822 var run_javascript_from_state = function (p) {
4823   var c = execution_ctx_initial(prog_intro_strictness(p));
4824   var tvoid _ = execution_ctx_binding_inst(Codetype_global, None, p, m_k_nil);
4825   return (run_prog(p));
4826 };
4827
4828 var run_javascript_from_result = function (w, p) {
4829   var %success_pat_any_5 = w;
4830   return (run_javascript_from_state(p));
4831 };
4832
4833 var run_javascript = function (p) {
4834   return (run_javascript_from_state(p));
```

Navigation Panel

Interpreter

Interpreter State

```
p: <synux-objects>
```

And Beyond

- extension to current version of JavaScript
 - ongoing, we now can debug it using jsexplain itself
 - engineer hired to work on this in September
- towards a typed specification?
 - PR 1135: Explicitly note mathematical values
 - Issue 496: abstract operations don't always return Completion Records
- better trace navigator
 - links to the specification

- needed to prove invariants of the specification
- modular description of the semantics with a simpler induction principle
 - POC for a small language
- we're hiring!¹ (For postdoctoral and PhD positions.)

¹<https://jobs.inria.fr/public/classic/en/offres/2018-00432>

Generalization to Other Languages

- MLExpain²
- plans to do it for Hop.js
- framework to describe semantics

²<https://github.com/Docteur-Lalla/mlexplain/tree/mlexplain>