Sémantique Formelle de JavaScript

Les Enjeux du Passage à l'Échelle

Alan Schmitt

9 janvier 2020

Motivation

How do you trust your software?

How do you trust your software?

- Manual verification
  - does not scale

manual verification

How do you trust your software?

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

manual verification bug finders

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

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

How do you trust the tool that verifies your software?

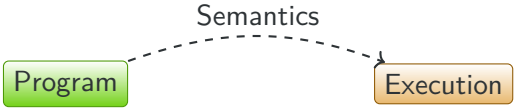
- Manual verification
  - does not scale
- Automatic bug finder
  - may miss some bugs
- Automatic, sound verifier
  - show the absence of bugs, may raise false alarms
    ex: the Astrée static analyzer
- Formally-verified verifier
  - the verifier comes with a soundness proof
    that is machine checked

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

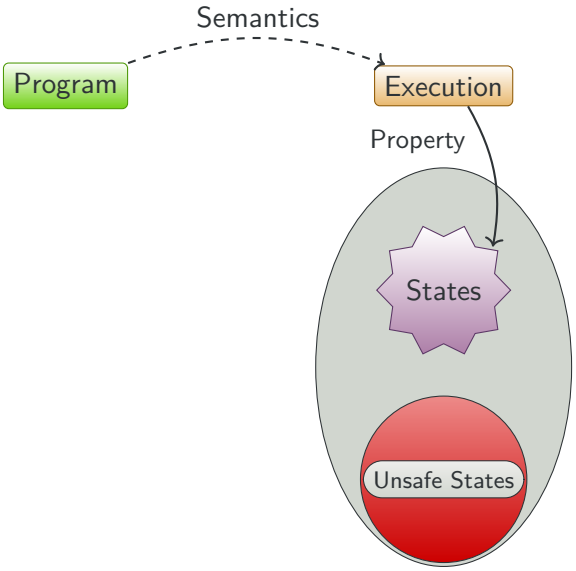
"Air France Airbus A380-800 F" by BriYYZ originally posted to Flickr as Arriving LAX north complex, credit BriYYZ, link

manual verification | bug finders | sound verifiers | verified verifiers

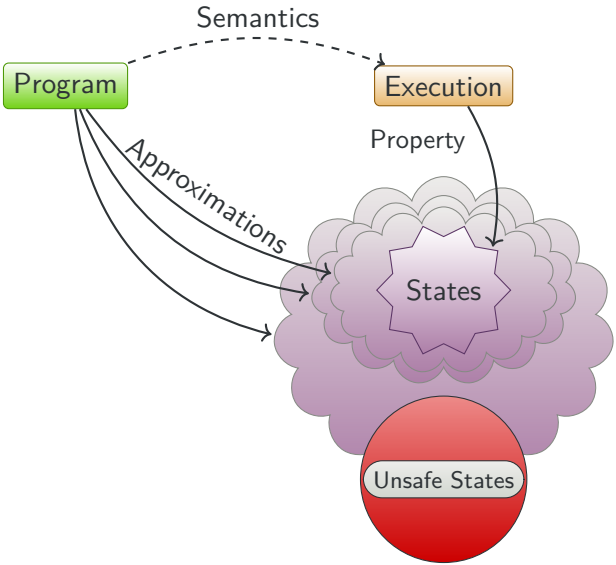
Certified Analyses



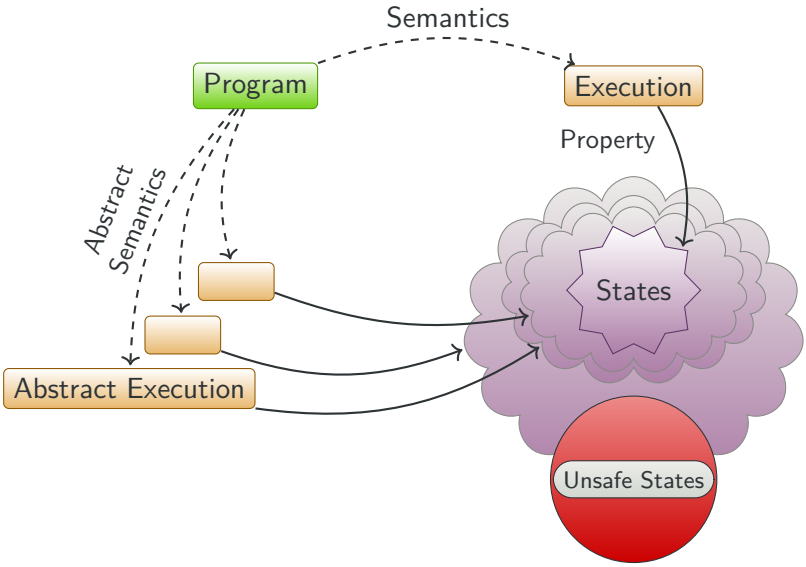
Certified Analyses



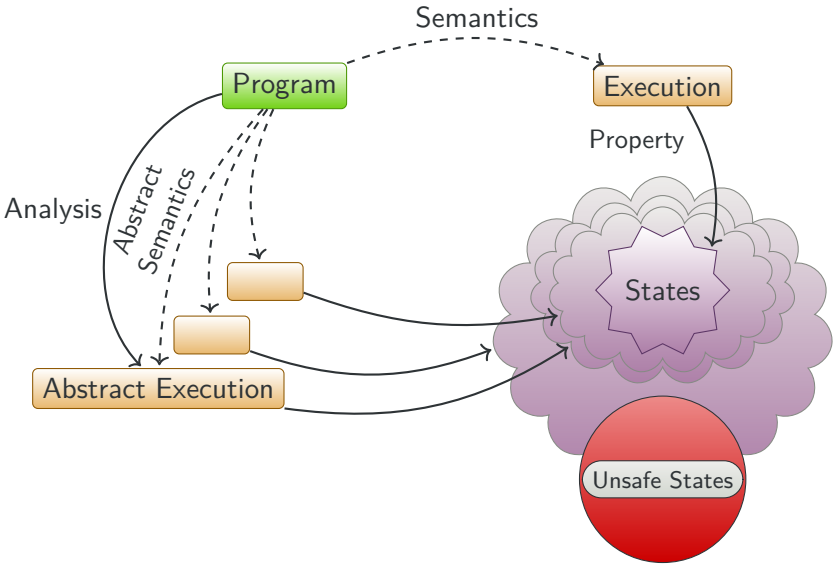
Certified Analyses



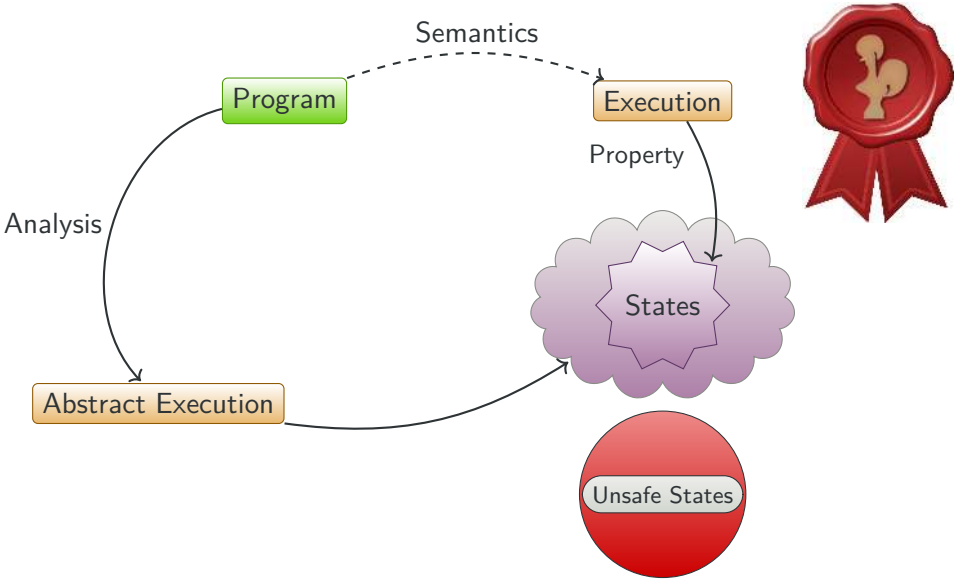
Certified Analyses



Certified Analyses



Certified Analyses



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

JavaScript is Everywhere

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

World Wide Web

The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
Pointers to the world's online information, subjects , W3 servers, etc.
Help
on the browser you are using
Software Products
A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
Details of protocols, formats, program internals etc
Bibliography
Paper documentation on W3 and references.
People
A list of some people involved in the project.
History
A summary of the history of the project.
How can I help ?
If you would like to support the web..
Getting code
Getting the code by anonymous FTP , etc.

credit Brian Solis

JavaScript under the Hood

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Test Button</title>
    </head>
    <body>
        <div id="div">0</div>
        <button onclick="incr()">+1</button>

        <script type="text/javascript">
         var x = 0;
         const div = document.getElementById("div")
         function incr(){
             x = x + 1;
             div.replaceChild(document.createTextNode(x), div.childNodes[0])
         }
        </script>

    </body>
</html>
```

JavaScript under the Hood

```
<!doctype html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Test Button</title>
    </head>
    <body>
        <div id="div">0</div>
        <button onclick="incr()">+1</button>

        <script type="text/javascript">
         var x = 0;
         const div = document.getElementById("div")
         function incr(){
             x = x + 1;
             div.replaceChild(document.createTextNode(x), div.childNo
         }
        </script>

    </body>
</html>
```

https://www.decisionproblem.com/paperclips/index2.html

Rich, Interactive Web Pages

diaspora*

JavaScript and Teaching

https://sicp.comp.nus.edu.sg/

You are reading the textbook Structure and Interpretation of Computer Programs *by Harold Abelson and Gerald Sussman—with a twist. The textbook emphasizes the importance of abstraction for managing complexity, and introduces the reader to a host of concepts that lie at the heart of the field of computer science. Most of these ideas are independent of the programming language used to express them to employ actual computers for solving computational problems. They are programming-language independent. The twist then consists of replacing the programming language that is used in the examples. While the authors had used the programming language Scheme, this adaptation uses the language JavaScript.*

JavaScript and Web Security

A Dangerous Situation

A Dangerous Situation

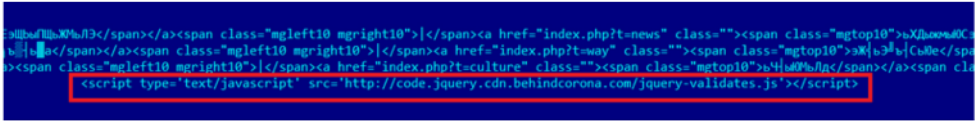
Kaspersky a découvert une faille zero-day utilisée pour lancer des attaques via Google Chrome

Les internautes sont invités à effectuer une mise à jour de leur navigateur

Le 4 novembre 2019 à 10:21, par Stéphane le calme | **0 commentaire**

Détails techniques

L'attaque utilise une injection semblable à une attaque de point d'eau sur un portail d'informations en coréen. Un code JavaScript malveillant a été inséré dans la page principale, ce qui charge à son tour un script de profilage à partir d'un site distant. Une attaque par point d'eau fait référence à un prédateur qui, au lieu d'aller chercher sa proie, préfère l'attendre à un endroit où il est sûr qu'elle viendra (en l'occurrence à un point d'eau pour s'abreuver).



Redirection vers la page de destination de l'exploit

source

JavaScript is Complex

Imperative and Functional

Variables

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

$\Rightarrow$ 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)
```

$\Rightarrow$ f(g,x) = 42

Objects

Literal Objects

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

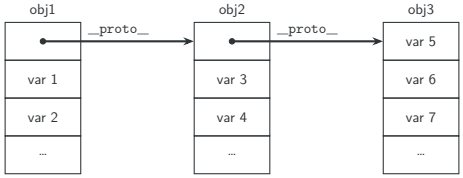
$\Rightarrow$ 1

Functions as Object Factories

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

$\Rightarrow$ 42

A Language Without Class



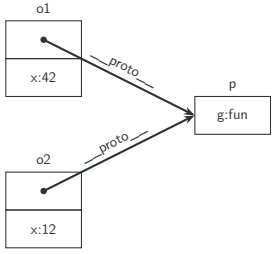
Prototypes and Object Factories

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

var o1 = new f(42); var o2 = new f(12)

console.log("o1.g() = " + o1.g() + ", o2.g() = " + o2.g())
```

$\Rightarrow$ `o1.g() = 43, o2.g() = 13`



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`

(![]+[])[+!+[]] 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, 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

$\Rightarrow$ strong need for standardization

A quick history of JavaScript and ECMAScript

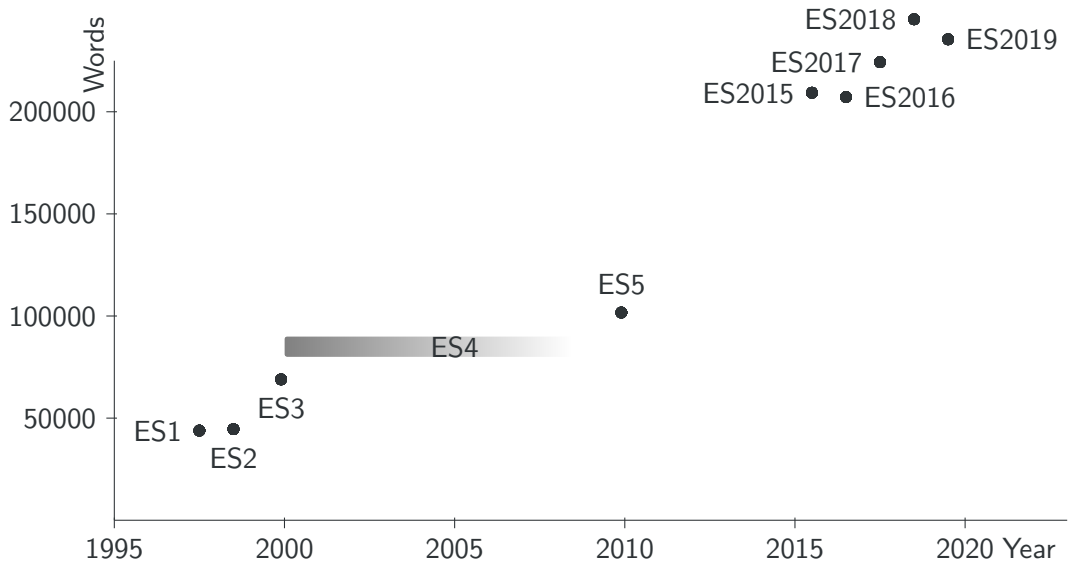
1995	Brendan Eich hired by Netscape to embed Scheme
May 1995	as Java is included in Netscape, scripting should have a similar syntax; JavaScript prototype developed in 10 days
Dec. 1995	JavaScript deployed in Netscape Navigator 2.0 beta 3
Aug. 1996	JScript deployed in Internet Explorer 3.0

code is supposed to run identically in every browser

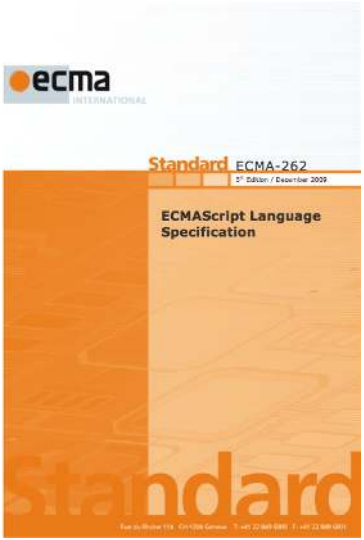
$\Rightarrow$ strong need for standardization

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



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

The specification

https://github.com/tc39/proposals

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

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

$\Rightarrow$

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

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]]().

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.

JavaScript, the Formalization

JavaScript Formalizations

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.

Semantics of While

1. Let $V =$ empty.

$$\frac{S, C, \texttt{while}_1(e_1, t_2, \texttt{empty}) \Downarrow o}{S, C, \texttt{while}(e_1, t_2) \Downarrow o}$$

Pretty-Big-Step Semantics

Arthur Charguéraud

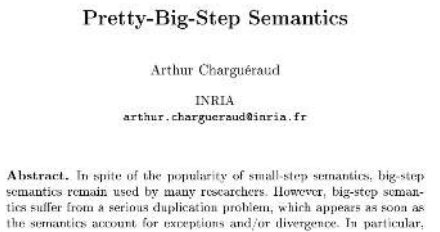
INRIA
arthur.charguer aud@inria.fr

Abstract. In spite of the popularity of small-step semantics, big-step semantics remain used by many researchers. However, big-step semantics suffer from a serious duplication problem, which appears as soon as the semantics account for exceptions and/or divergence. In particular,

Semantics of While

1. Let $V =$ empty.

$$\frac{S, C, \texttt{while}_1(e_1, t_2, \texttt{empty}) \Downarrow o}{S, C, \texttt{while}(e_1, t_2) \Downarrow o}$$



Pretty-Big-Step Semantics

Arthur Charguéraud

INRIA
arthur.chargueraud@inria.fr

Abstract. In spite of the popularity of small-step semantics, big-step semantics remain used by many researchers. However, big-step semantics suffer from a serious duplication problem, which appears as soon as the semantics account for exceptions and/or divergence. In particular,

```
(* 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*).

$$\frac{S, C, e_1 \Downarrow v_1 \qquad \texttt{to\_boolean}(v_1) = b \qquad S, C, \texttt{while}_2(e_1, t_2, rv, b) \Downarrow o}{S, C, \texttt{while}_1(e_1, t_2, rv) \Downarrow o}$$

$$\frac{}{S, C, \texttt{while}_2(e_1, t_2, rv, \texttt{false}) \Downarrow (S, rv)}$$

```
(* 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 o1 o,
    red_spec S C (spec_expr_get_value_conv spec_to_boolean e1) o1 ->
    red_stat S C (stat_while_2 labs e1 t2 rv o1) 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

JSCert

JSRef

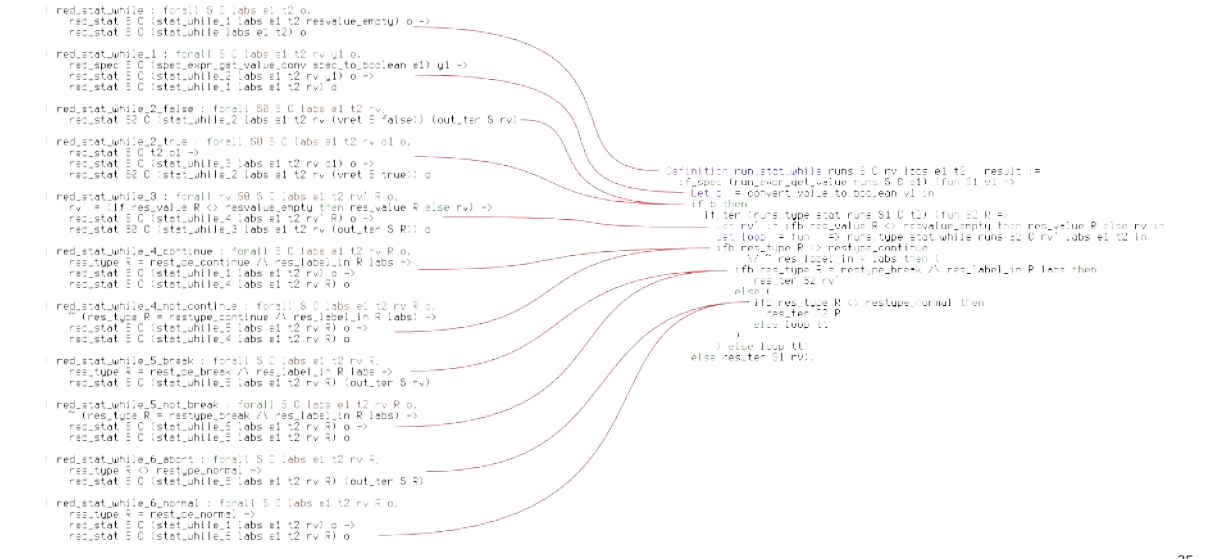
Coq

¬ Coq

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).
```

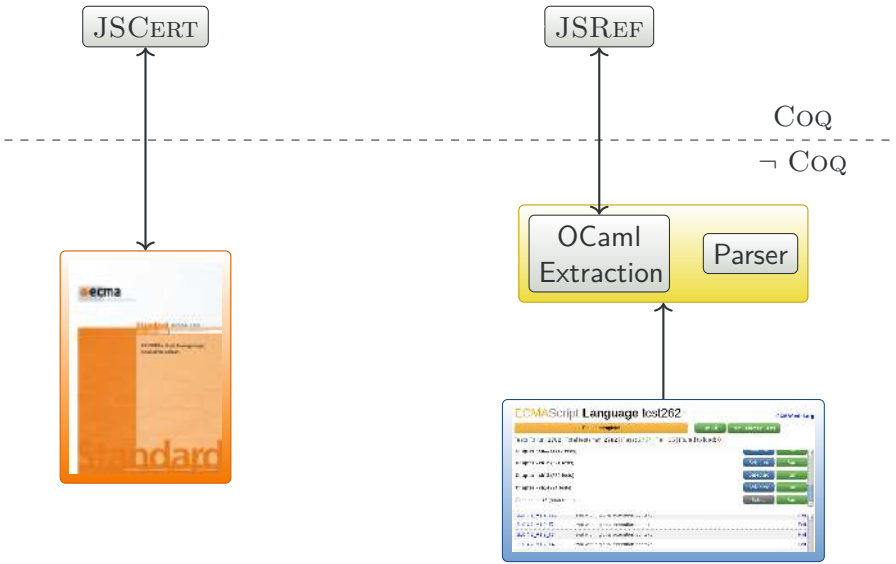
Fairly Close to the Inductive Rules



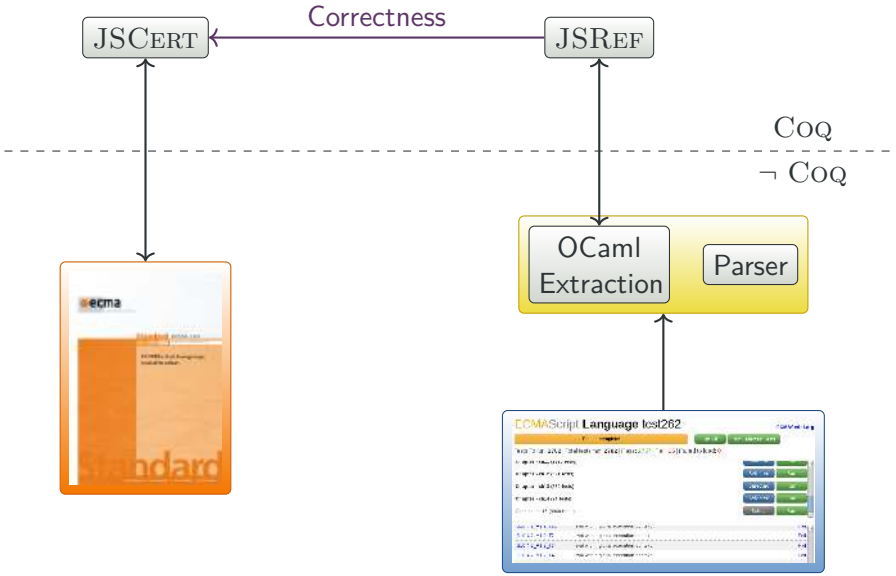
Extracted to OCaml

```
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)))
```

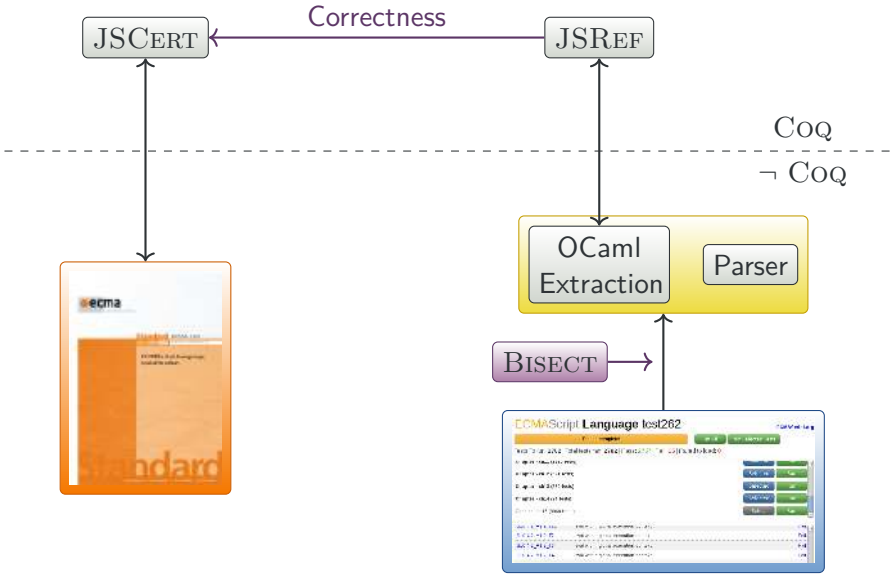
JavaScript Formalizations



JavaScript Formalizations

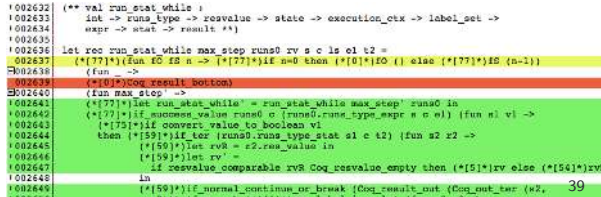


JavaScript Formalizations



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



JSCert (POPL 2014)

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







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

```
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

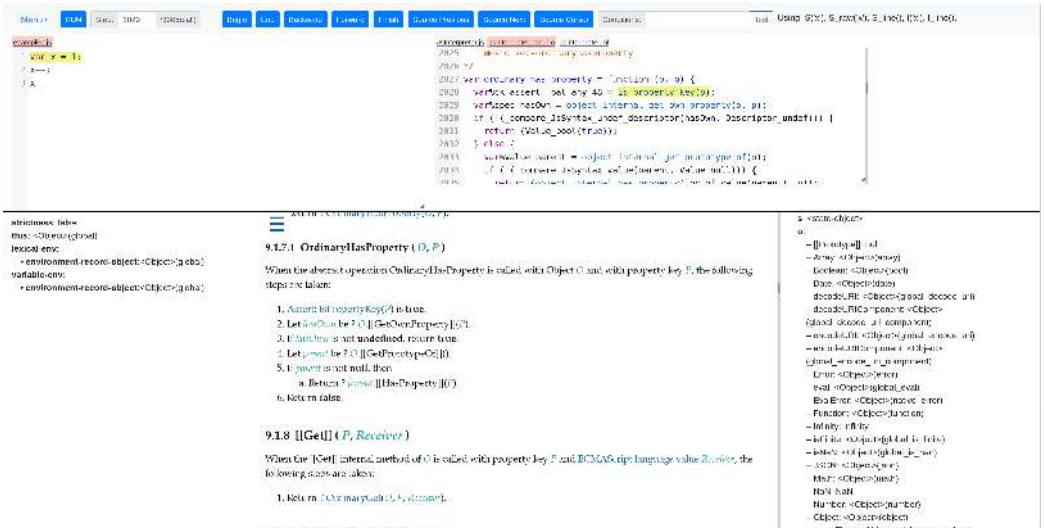
- motivation: run it in a browser
- uses compiler-libs to generate a typed AST, which we translate
- target is a tiny subset of JS
  - functions, objects (no prototype), arrays, string, numbers
  - no type conversion

```
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 { ... }})); }));
};
```

JSExplain (WWW 2018)

- 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

Demo



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

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

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

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

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

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

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

Recursion

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

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.

Recursion

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

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

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

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

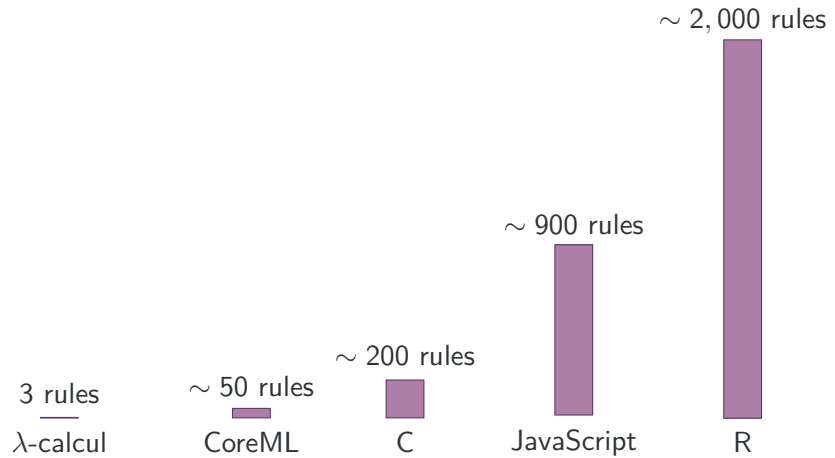
$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{tt} \qquad \sigma, s1 \Downarrow o}{\sigma, \texttt{if } e \texttt{ then } s1 \texttt{ else } s2 \Downarrow o}$$

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

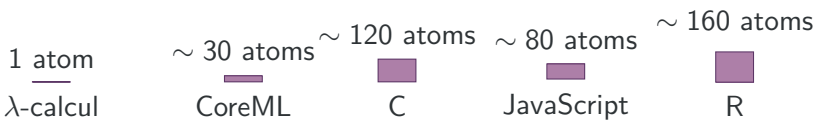
Evaluate `if e then s1 else s2` in state σ

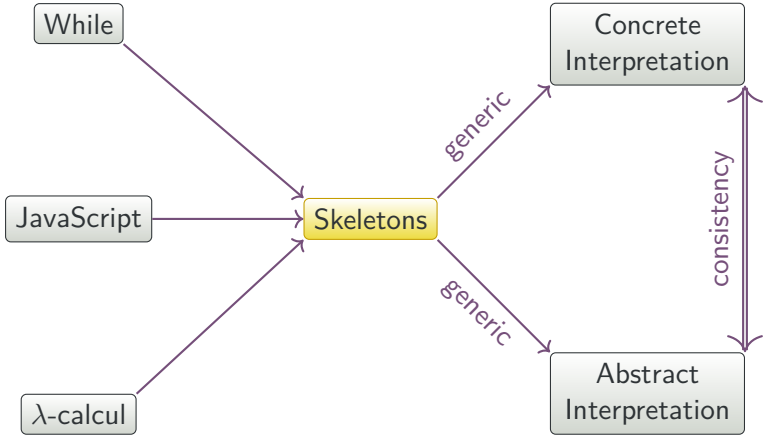
1. Let `v` the result of evaluating `e` in state σ.
2. If `v` is true, let `o` the result of evaluating `s1` in state σ.
3. If `v` is false, let `o` the result of evaluating `s2` in state σ.
4. Return `o`.

Size of Semantics (Number of Rules)



Size of Semantics (Number of Atoms)




While
JavaScript
λ-calcul
Skeletons
generic
generic
Concrete Interpretation
Abstract Interpretation
consistency

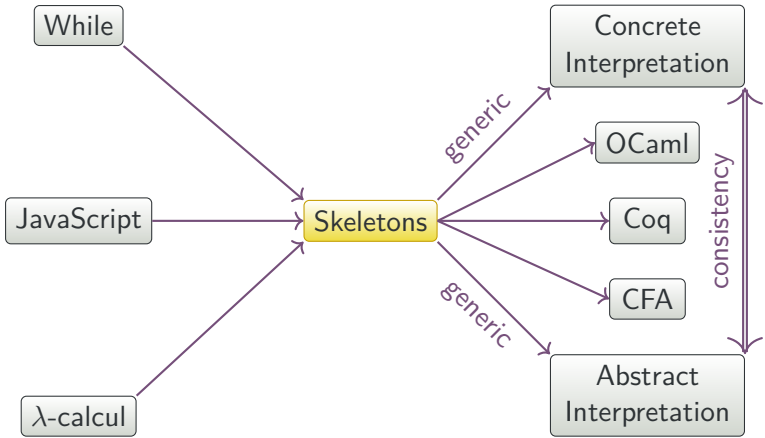
(POPL 2019)

Skeleton Example

```
rule eval_stmt (s, If (t1, t2, t3)) =
  let f1  = eval_expr (s, t1) in   (* recursion, sequence *)
  branch                           (* choice *)
    isTrue (f1) ;                  (* atom *)
    eval_stmt (s, t2)
  or
    isFalse (f1) ;
    eval_stmt (s, t3)
  end
```

Goals of Skeletal Semantics

- Simple framework capturing the structure of semantics
- Syntax close to algorithmic semantics and rule-based semantics
- Generic definition of interpretations
- Proof techniques to relate interpretations


While
JavaScript
λ-calcul
Skeletons
generic
generic
Concrete Interpretation
OCaml
Coq
CFA
Abstract Interpretation
consistency

Conclusion

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/