

# A Certified JAVAScript Interpreter

MARTIN BODIN    ALAN SCHMITT

6 Février 2013

JFLA 2013, AUSOIS

# En une planche

- ❶ Le « langage assembleur d'Internet »,
- ❷ Un langage *dynamique*,
- ❸ Mais une norme précise : ECMA SCRIPT 3–5,
- ❹ Un comportement mélangeant les fonctionnalités de tous les autres langages de programmation :
  - ordre supérieur,
  - effets de bords et variables globales,
  - prototypes pouvant simuler des classes.



`{}` + `{}`

```
{ } + { }
```

```
→ NaN
```

```
{ } + [ ]
```

```
{ } + { }
```

```
→ NaN
```

```
{ } + [ ]
```

```
→ 0
```

```
[ ] + { }
```

```
{ } + { }
```

```
→ NaN
```

```
{ } + [ ]
```

```
→ 0
```

```
[ ] + { }
```

```
→ "[object Object]"
```

```
( { } + { } )
```

```
{ } + { }
```

```
→ NaN
```

```
{ } + [ ]
```

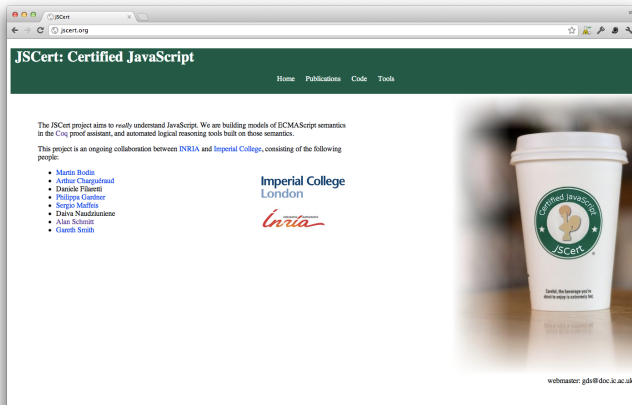
```
→ 0
```

```
[ ] + { }
```

```
→ "[object Object]"
```

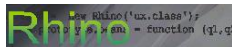
```
( { } + { } )
```

```
→ "[object Object][object Object]"
```

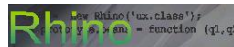


- Logique de séparation,
- Analyses statiques,
- Preuves de correction de compilateurs vers JAVASCRIPT...





...

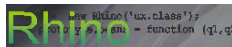


⋮

```
1  "a" ; while(true) { try{ "b" } finally { break } }
```

"a"

"b"

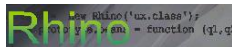


...

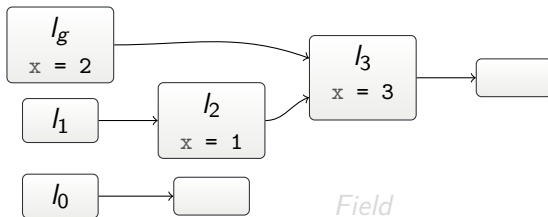
Mêmes  
abstractions  
que la  
spécification

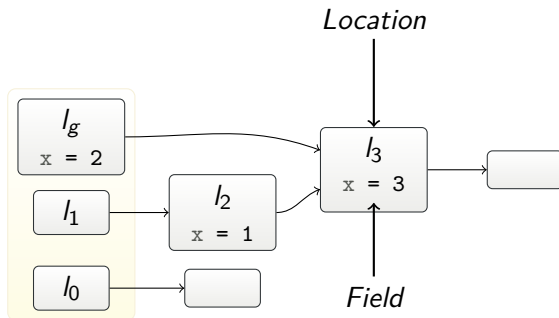


Testés sur  
les mêmes  
suites

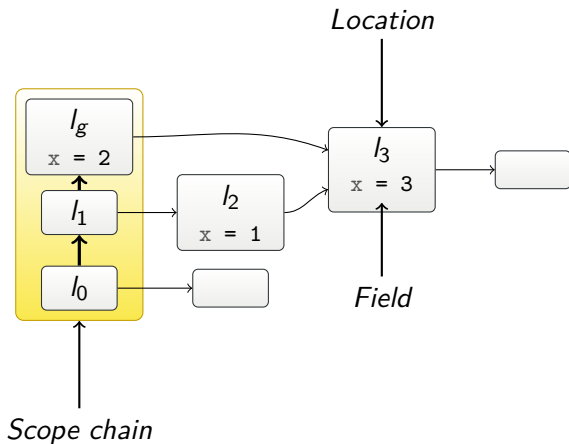


```
1  var Common = { x : 1 }
2  var Class = function (){}
3  Class.prototype = Common
4
5  var x = 0
6  var obj = new Class
7
8  function pr () {
9      with (obj) {
10         console.log("x: " + x + ", Common.x: " +
11                     Common.x)
12     }
13
14     pr () // "x: 1, Common.x: 1"
15     Common.x = 2 ; pr () // "x: 2, Common.x: 2"
16     with (obj){ x = 3 } ; pr () // "x: 3, Common.x: 2"
17     Common.x = 4 ; pr () // "x: 3, Common.x: 4"
```

*Location**Scope chain*

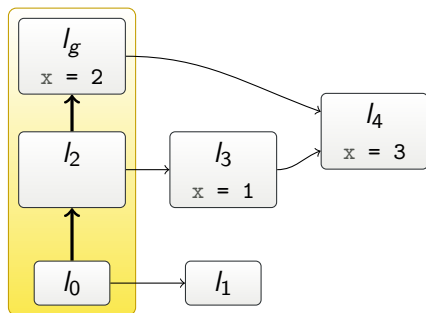


*Scope chain*





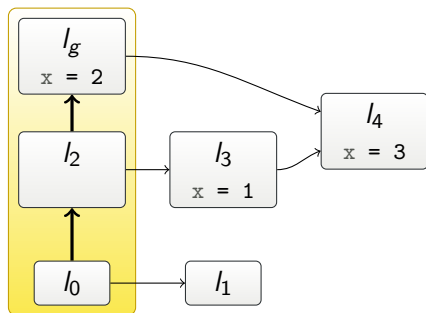
Chaque location possède un prototype implicite `@proto`.



*Scope chain*

# Lecture

Chaque location possède un prototype implicite *@proto*.

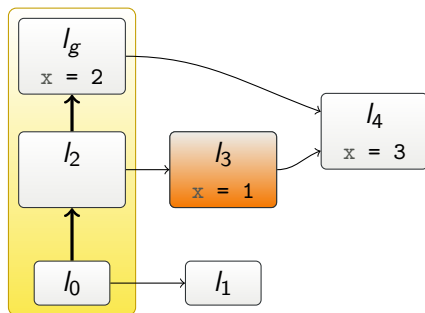


*Scope chain*

```
1 var y = x ;
```

# Lecture

Chaque location possède un prototype implicite `@proto`.

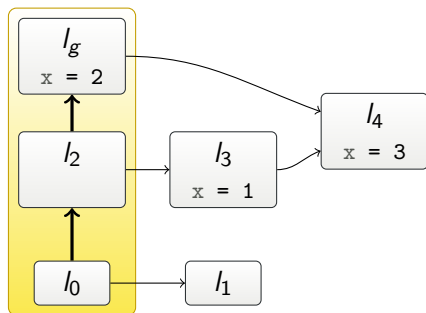


*Scope chain*

```
1 var y = x ;
```

# Écriture

Chaque location possède un prototype implicite `@proto`.

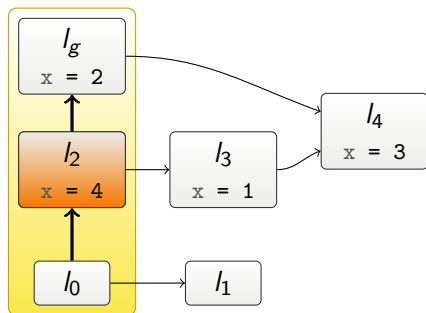


*Scope chain*

```
1 x = 4 ;
```

# Écriture

Chaque location possède un prototype implicite `@proto`.



*Scope chain*

```
1 x = 4 ;
```

# La formalisation COQ de la sémantique

- Basée (entre autres) sur une formalisation de SERGIO MAFFEIS.
- Utilise des abstractions similaire à celles de la spécification ECMA SCRIPT 3, et bientôt ECMA SCRIPT 5.

**Operational semantics:**  $H, L, e \longrightarrow H', r$ .

Notation:  $H, L, e \xrightarrow{\gamma} H', v \triangleq \exists r. (H, L, e \longrightarrow H', r \wedge \gamma(H', r) = v)$ .

(Definition)

$H, L, e \longrightarrow H', v$

(Value)

$H, L, v \longrightarrow H, v$

$H, L, \text{var } e \longrightarrow H', \text{undefined}$

(Member Access)

$H, L, e \xrightarrow{\gamma} H', l$

$l \neq \text{null}$

$H, L, e.x \longrightarrow H', l.x$

(Computed Access)

$H, L, e1 \xrightarrow{\gamma} H_1, l$

$l \neq \text{null}$

$H_1, L, e2 \xrightarrow{\gamma} H', x$

$H, L, e1[e2] \longrightarrow H', l.x$

(Variable)

$\sigma(H, L, x) = l$

$H, L, x \longrightarrow H, l.x$

(Object)

$H_0 = H * \text{obj}(l, l_{op})$

$\forall i \in 1..n. \left( \begin{array}{l} H_{i-1}, L, e1 \xrightarrow{\gamma} H'_i, v_i \\ H_i = H'_i[(l, x1) \mapsto v_i] \end{array} \right)$

$H, L, \{x1:e1, \dots, xn:en\} \longrightarrow H_n, l$

Fichier	Taille (Kio)		Description
	Définitions	Preuves	
JsSyntax.v	4.5	0	Syntaxe de JAVASCRIPT
JsSyntaxAux.v	1.5	7.5	Propriétés basiques des objets définis dans JsSyntax.v
JsSemantic.v	21.5	0	Règles de réduction de JAVASCRIPT
JsSemanticAux.v	0.5	15.5	Propriétés basiques des objets définis dans JsSemantic.v
JsWf.v	4.5	0	Définition des invariants nécessaires pour que les règles aient un sens
JsWfAux.v	0	6.5	Propriétés basiques des objets définis dans JsWf.v
JsSafety.v	0	33.5	Preuve de la conservation des invariants de JsWf.v lors de l'exécution d'un programme JAVASCRIPT
JsScopes.v	0	1.5	Quelques lemmes pour l'interpréteur
JsInterpreter.v	17	1.5	Définition de l'interpréteur JAVASCRIPT
JsInterpreterProof.v	0.5	42.5	Preuve de la correction de l'interpréteur
<b>Total</b>	<b>50</b>	<b>108.5</b>	

Fichier	Taille (Kio)		Description
	Définitions	Preuves	
JsSyntax.v	4.5	0	Syntaxe de JAVASCRIPT
JsSyntaxAux.v	1.5	7.5	Propriétés basiques des objets définis dans JsSyntax.v
JsSemantic.v	21.5	0	Règles de réduction de JAVASCRIPT
JsSemanticAux.v	0.5	15.5	Propriétés basiques des objets définis dans JsSemantic.v
JsWf.v	4.5	0	Définition des invariants nécessaires pour que les règles aient un sens
JsWfAux.v	0	6.5	Propriétés basiques des objets définis dans JsWf.v
JsSafety.v	0	33.5	Preuve de la conservation des invariants de JsWf.v lors de l'exécution d'un programme JAVASCRIPT
JsScopes.v	0	1.5	Quelques lemmes pour l'interpréteur
JsInterpreter.v	17	1.5	Définition de l'interpréteur JAVASCRIPT
JsInterpreterProof.v	0.5	42.5	Preuve de la correction de l'interpréteur
<b>Total</b>	<b>50</b>	<b>108.5</b>	
<b>Trusted base</b>	<b>30.5</b>	<b>0</b>	



```

1 Inductive red : heap → scope → expr →
2   heap → result → Prop :=
3   :
4   | red_assign : ∀ l f v h0 h1 h2 h3 s e1 e2 r2,
5     red h0 s e1 h1 (Ref l f) →
6     red h1 s e2 h2 r2 →
7     getval h2 r2 v →
8     h3 = update h2 l f v →
9     red h0 s (exp_assign e1 e2) h3 v

```

(Assignment)

$$\frac{
 \begin{array}{l}
 H, L, e1 \longrightarrow H_1, l \cdot x \\
 H_1, L, e2 \xrightarrow{\gamma} H_2, v \\
 H' = H_2[(l, x) \mapsto v]
 \end{array}
 }{
 H, L, e1 = e2 \longrightarrow H', v
 }$$

```
1 Inductive red : heap → scope → expr →  
2   heap → result → Prop :=  
   ⋮
```

```
1 Fixpoint run (max_step : nat) (h0 : heap) (s : scope)  
2   (e : expr) : out :=  
3   match max_step with  
4   | 0 ⇒ out_bottom  
5   | S max_step' ⇒  
   ⋮
```

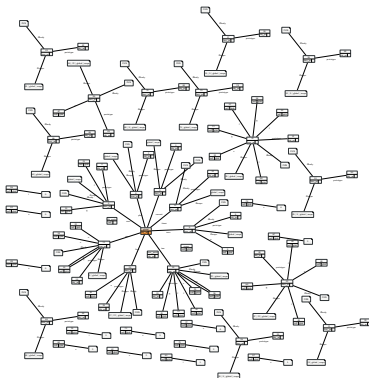
## Règle issue de la sémantique

```
1 | red_assign :  $\forall l f v h0 h1 h2 h3 s e1 e2 r2,$   
2   red h0 s e1 h1 (Ref l f)  $\rightarrow$   
3   red h1 s e2 h2 r2  $\rightarrow$   
4   getval h2 r2 v  $\rightarrow$   
5   h3 = update h2 l f v  $\rightarrow$   
6   red h0 s (exp_assign e1 e2) h3 v
```

## Réduction dans l'interpréteur


```
1 | exp_assign e1 e2  $\Rightarrow$   
2   if_success (run' h0 s e1) (fun h1 r1  $\Rightarrow$   
3     if_is_ref h1 r1 (fun l f  $\Rightarrow$   
4       if_success_value (run' h1 s e2) (fun h2 v  $\Rightarrow$   
5         out_return (update h2 l f v) v)))
```

# Ça compile !




Mais qu'en est-il de la correction ?

```
1 Theorem run_correct :  $\forall m\ h\ s\ e\ h'\ v,$   
2   run m h s e = out_return h' (ret_res v)  $\rightarrow$   
3   ok_heap h  $\rightarrow$   
4   ok_scope h s  $\rightarrow$   
5   red h s e h' v.
```



[What Is Ecma](#)
[Activities](#)
[News](#)
[Standards](#)



Rue du Rhône 114 CH-1204 Geneva  
 T: +41 22 849 6000 F: +41 22 849 6001

[SITE MAP](#)

[Standards Index](#)  
[Standards List](#)  
[Withdrawn Standards](#)  
[Tech. Reports Index](#)  
[Tech. Reports List](#)  
[Withdrawn Tech. Reports](#)  
[Mementos](#)

[Printer Friendly Version](#)  
[Back](#)

## Standard ECMA-262

### ECMAScript Language Specification

*Edition 5.1 (June 2011)*

---

The latest version of this Ecma Standard is available [here](#).

---

You can download the previous replaced "historical" edition of this Ecma Standard, free of charge:

File name	Content
<a href="#">ECMA-262 1st edition</a>	Acrobat (r) PDF file
<a href="#">ECMA-262 2nd edition</a>	Acrobat (r) PDF file
<a href="#">ECMA-262 3rd edition</a>	Acrobat (r) PDF file
ECMA-262 4th edition	NOT EXISTING*
<a href="#">ECMA-262 5th edition</a>	Acrobat (r) PDF file

\* Please note that for ECMAScript Edition 4 the Ecma standard number "ECMA-262 Edition 4" was reserved but not used in the Ecma publication process. Therefore "ECMA-262 Edition 4" as an Ecma International publication does not exist.

This ECMAScript Language Specification is also available in HTML file format [here](#).

Les choses ont un petit peu changé depuis le papier. . .

### 11.13.1 Simple Assignment ( = )

The production *AssignmentExpression* : *LeftHandSideExpression* = *AssignmentExpression* is evaluated as follows:

1. Let *lref* be the result of evaluating *LeftHandSideExpression*.
2. Let *rref* be the result of evaluating *AssignmentExpression*.
3. Let *rval* be GetValue(*rref*).
4. Throw a **SyntaxError** exception if the following conditions are all true:
  - Type(*lref*) is Reference is **true**
  - IsStrictReference(*lref*) is **true**
  - Type(GetBase(*lref*)) is Environment Record
  - GetReferencedName(*lref*) is either "**eval**" or "**arguments**"
5. Call PutValue(*lref*, *rval*).
6. Return *rval*.

**NOTE** When an assignment occurs within strict mode code, its *LeftHandSide* must not evaluate to an unresolvable reference. If it does a **ReferenceError** exception is thrown upon assignment. The *LeftHandSide* also may not be a reference to a data property with the attribute value `[[Writable]]:false`, to an accessor property with the attribute value `[[Set]]:undefined`, nor to a non-existent property of an object whose `[[Extensible]]` internal property has the value **false**. In these cases a **TypeError** exception is thrown.

(\*\* Assignment \*)

```
| red_expr_assign : ∀S C opo e1 e2 o o1,
  red_expr S C e1 o1 →
  red_expr S C (expr_assign_1 o1 opo e2) o →
  red_expr S C (expr_assign e1 opo e2) o
```

```
| red_expr_assign_1_simple : ∀S0 S C rv e2 o o1,
  red_expr S C (spec_expr_get_value e2) o1 →
  red_expr S C (expr_assign_4 rv o1) o →
  red_expr S0 C (expr_assign_1 (out_ter S rv) None e2) o
```

```
| red_expr_assign_1_compound : ∀S0 S C rv op e2 o o1,
  red_expr S C (spec_get_value rv) o1 →
  red_expr S C (expr_assign_2 rv o1 op e2) o →
  red_expr S0 C (expr_assign_1 (out_ter S rv) (Some op) e2) o
```

```
| red_expr_assign_2_compound_get_value : ∀S0 S C rv op v1
  o2 e2 o,
  red_expr S C (spec_expr_get_value e2) o2 →
  red_expr S C (expr_assign_3 rv v1 op o2) o →
  red_expr S0 C (expr_assign_2 rv (out_ter S v1) op e2) o
```

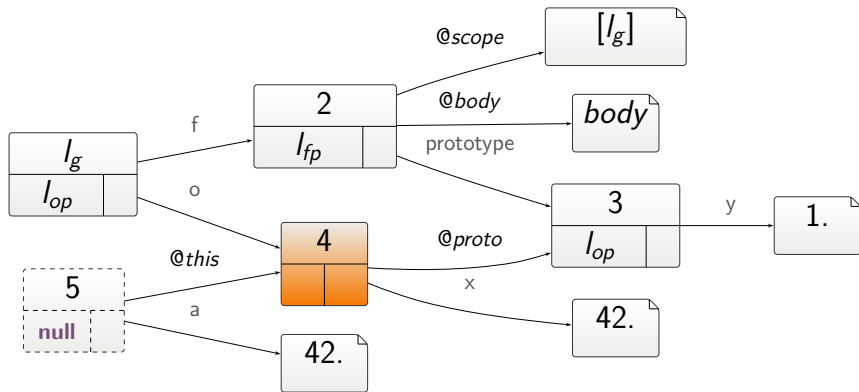
```
| red_expr_assign_3_compound_op : ∀S0 S C rv op v1 v2 o1
  o,
  red_expr S C (expr_binary_op_3 op v1 v2) o1 →
  red_expr S C (expr_assign_4 rv o1) o →
  red_expr S0 C (expr_assign_3 rv v1 op (out_ter S v2)) o
```

```
| red_expr_assign_4_put_value : ∀S0 S C rv v o1 o,
  red_expr S C (spec_put_value rv v) o1 →
  red_expr S C (expr_assign_5 v o1) o →
  red_expr S0 C (expr_assign_4 rv (out_ter S v)) o
```

```
| red_expr_assign_5_return : ∀S0 S C rv' v,
  red_expr S0 C (expr_assign_5 v (out_ter S rv')) (
    out_ter S v)
```



```
1  | expr_assign e1 opo e2 ⇒
2    if_success (run_expr' S C e1) (fun S1 rv1 ⇒
3      let follow S rv' :=
4        match rv' with
5        | resvalue_value v ⇒
6          if_success (ref_put_value run_call' S C rv1 v) (fun S' rv2 ⇒
7            out_ter S' v)
8        | _ ⇒ result_stuck
9      end in
10     match opo with
11     | None ⇒
12       if_success_value (run_expr' S1 C e2) follow
13     | Some op ⇒
14       if_success_value (out_ter S1 rv1) (fun S2 v1 ⇒
15         if_success_value (run_expr' S2 C e2) (fun S3 v2 ⇒
16           if_success (run_binary_op max_step' run_call' S3 C op v1
17             v2) follow))
18     end)
```



Avez-vous des questions ?

- 1 À propos de JAVASCRIPT
- 2 JSCert ?
- 3 Vue d'ensemble de la sémantique de JAVASCRIPT
- 4 Un interpréteur prouvé correct
- 5 Le passage à ECMASCRIPT 5

```

(* assign *)
forwards [(?&?) | (r1&h1&eq1)]: elim_if_success R; tryfalse.
rewrite eq1 in R. simpl in R.
forwards [(eqw&v'&eqv') | (l&f&eq')]: elim_if_is_ref R.
lets (h'0&eqw'): eqw. forwards*: wrong_not_ret eqw'.
rewrite eq' in R. simpl in R.
forwards [(?&?) | [(v0&h2&eq2&eqv0) | (v0&h2&b&eq2&eqv0)]]:
  elim_if_success_value R; tryfalse.
rewrite eq2 in R. simpls. rewrite eqv0 in R. simpls.
  forwards*: wrong_not_ret R.
rewrite eq2 in R. simpls. rewrite eqv0 in R. simpls.
inverts* R.
forwards* R1: run_correct eq1.
forwards* (OK1&OKL1&OKr1): sub_safety R1.
inverts* OKr1; tryfalse.
inverts H0.
forwards* R2: run_correct eq2.
forwards* (OK2&OKL2&OKr2): sub_safety R2.
apply* red_assign.
apply* getvalue_comp_correct.

```

Destruction de  
 la définition de  
 l'interpréteur

Preuve que les termes  
 intermédiaires sont bien  
 formés

Reconstruction des règles  
 de dérivation

