Formalisation de Langages de programmation en Coq

Martin Bodin

14 mars

martin.bodin@inria.fr

Parcours

- 2009–2013 : ENS, Lyon.
- 2013–2016 : Doctorat, Inria, Rennes.
  JavaScript en Coq 
- 2017–2018 : Postdoc, CMM, Santiago de Chile.
  R en Coq 
- 2018–2020 : Postdoc, Imperial College, Londres.
  WebAssembly en Coq 
  Squelettes
- 2020– : Chargé de recherche, Inria, Grenoble.
  Monades pour la formalisation de langages
  Coq  dans l'enseignement en mathématiques

Parcours

- 2009–2013 : ENS, Lyon.
- 2013–2016 : Doctorat, Inria, Rennes.
  JavaScript en Coq 
- 2017–2018 : Postdoc, CMM, Santiago de Chile.
  R en Coq 
- 2018–2020 : Postdoc, Imperial College, Londres.
  WebAssembly en Coq 
  Squelettes
- 2020– : Chargé de recherche, Inria, Grenoble.
  Monades pour la formalisation de langages
  Coq  dans l'enseignement en mathématiques

Formalisations de langages

Outils pour les utiliser

Systèmes embarqués reconfigurables

Systèmes embarqués reconfigurables

Conception et modèles de programmation

- Modularité : pouvoir modifier des composants de manière dynamique,
- Consilier expressivité et l'analysabilité statique.

Systèmes embarqués reconfigurables

Conception et modèles de programmation

- Modularité : pouvoir modifier des composants de manière dynamique,
- Consilier expressivité et l'analysabilité statique.

Analyses et programmation de systèmes temps-réel

- Bibliothèque Prosa pour l'ordonnancabilité,
- Ordonnancement multi-critère : temps et énergie.

Systèmes embarqués reconfigurables

Conception et modèles de programmation

- Modularité : pouvoir modifier des composants de manière dynamique,
- Consilier expressivité et l'analysabilité statique.

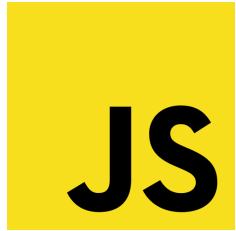
Analyses et programmation de systèmes temps-réel

- Bibliothèque Prosa pour l'ordonnancabilité,
- Ordonnancement multi-critère : temps et énergie.

Gestion d'erreurs et analyse causale

- Sémantiques réversibles,
- Explicabilité causale d'un système.

Le monde est compliqué



- Premier langage sur Github,
- Présent sur 95 % des sites web,
- Utilisé dans de nombreuses interfaces.

- `![]` ⇝ **`false`**
- `+![]` ⇝ `+`**`false`** ⇝ `0`
- `![]+[]` ⇝ **`false`**`+[]` ⇝ `"false"`
- `(![]+[])[+![]]` ⇝ `"false"[0]` ⇝ `"f"`

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

- `![]` ⇝ **`false`**
- `+![]` ⇝ `+`**`false`** ⇝ `0`
- `![]+[]` ⇝ **`false`**`+[]` ⇝ `"false"`
- `(![]+[])[+![]]` ⇝ `"false"[0]` ⇝ `"f"`

```
(!![]+[])[!+[]+!+[]+!+[]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]
 +(+[![]]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]
  +(!![]+[])[+[]]])[+!+[]+[+!+[]]]
 +([![]]+[][[]])[+!+[]+[+[]]]+([][[]]+[])[+!+[]]+(![]+[])[+[]]
 +(!![]+[][(![]+[])[+[]]+(![]+[])[!+[]+!+[]]+(![]+[])[+!+[]]
  +(!![]+[])[+[]]])[+!+[]+[+[]]]
⇝ "ens info"
```

- `![]` ⇝ `false`
- `+![]` ⇝ `+false` ⇝ `0`
- `![]+[]` ⇝ `false+[]` ⇝ `"false"`
- `(![]+[])[+![]]` ⇝ `"false"[0]` ⇝ `"f"`

```
1  (!![]+[])[!+[]+!+[]+!+[]]+([][[]]+[])[+!+[]]+(![]+[])[!+[]+!+[]+!+[]]
2   +(+[!![]+[][(!
3    +(!![]+[])[+[
4   +([![]]+[][[]])
5   +(!![]+[][(![]+
6    +(!![]+[])[+[]
7  ⇝ "ens info"
```

- Premier[1] langage de traitement de données :
  - Simulations physiques,
  - Données linguistiques,
  - Données de santé,
  - etc.

[1] Sauf si on compte les feuilles de calcul.

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
v[-2]                         # Retourne c(11, 13, 14, 15)
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
v[-2]                         # Retourne c(11, 13, 14, 15)
v[-indices]                   # Retourne c(12, 14)
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
v[-2]                         # Retourne c(11, 13, 14, 15)
v[-indices]                   # Retourne c(12, 14)
v[c(0, 2.9)]                  # Retourne 12
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
v[-2]                         # Retourne c(11, 13, 14, 15)
v[-indices]                   # Retourne c(12, 14)
v[c(0, 2.9)]                  # Retourne 12
v[c(FALSE, TRUE, FALSE)]      # Retourne c(12, 15)
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
v[-2]                         # Retourne c(11, 13, 14, 15)
v[-indices]                   # Retourne c(12, 14)
v[c(0, 2.9)]                  # Retourne 12
v[c(FALSE, TRUE, FALSE)]      # Retourne c(12, 15)
v["a"]                        # Retourne NA
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
v[-2]                         # Retourne c(11, 13, 14, 15)
v[-indices]                   # Retourne c(12, 14)
v[c(0, 2.9)]                  # Retourne 12
v[c(FALSE, TRUE, FALSE)]      # Retourne c(12, 15)
v["a"]                        # Retourne NA
f <- function(a, b)
       v[a + b]               # ??
```

```r
v <- c(11, 12, 13, 14, 15)
v[1]                          # Retourne 11
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(13, 15, 11)
v[-2]                         # Retourne c(11, 13, 14, 15)
v[-indices]                   # Retourne c(12, 14)
v[c(0, 2.9)]                  # Retourne 12
v[c(FALSE, TRUE, FALSE)]      # Retourne c(12, 15)
v["a"]                        # Retourne NA
f <- function(a, b)
       v[a + b]               # ??
'[' <- function (a, b) 42
v[indices]                    # Retourne 42
```

Opinion

- Les langages de programmation **populaires** sont **complexes**,
- Leur complexité est une **source d'erreurs**,
- Il y a ainsi un besoin d'**analyse** et d'**outils** pour ces langages,
- Les langages sont complexes $\implies$ le code des outils est complexe,
  $\implies$ **erreur dans l'outil** probable,
  $\implies$ crise de la **confiance**.
- L'**assistant de preuve Coq**  peut aider à résoudre cette crise.

Approche alternative : des langages de programmation plus réguliers

Mais on ne va pas en parler aujourd'hui

typescriptlang.org

julialang.org

Formaliser en Coq  des languages complexes

Servir de **base de confiance** à des analyses/compilations certifiées.

$\Longrightarrow$ Prendre le langage tel qu'il est, sans simplification.

Formaliser en Coq des languages complexes

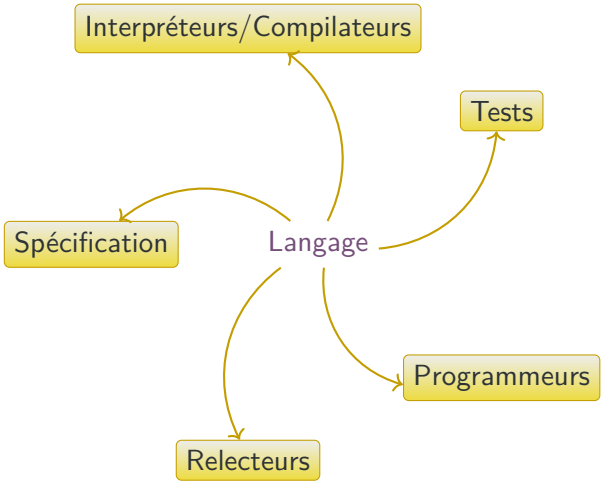
Servir de **base de confiance** à des ~~analyses/compilations~~ certifiées.

$\Longrightarrow$ Prendre le langage tel qu'il est, sans simplification.

Formalisation très large...
Comment lui faire confiance ?

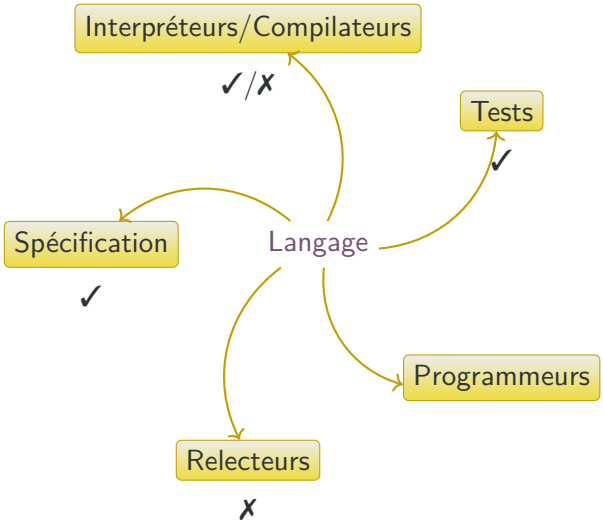
Quelle est la définition d'un langage ?

Cela dépend à qui on demande.



Quelle est la définition d'un langage ?

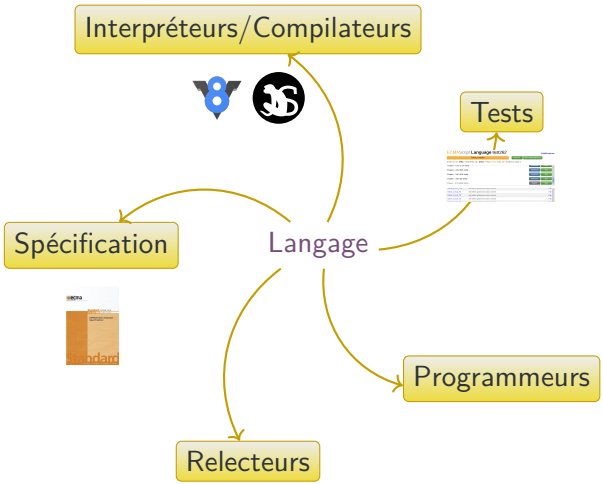
Cela dépend à qui on demande.




```
1  register n = (count + 7) / 8;
2  switch (count % 8) {
3  case 0: do { *to = *from++;
4  case 7:      *to = *from++;
5  case 6:      *to = *from++;
6  case 5:      *to = *from++;
7  case 4:      *to = *from++;
8  case 3:      *to = *from++;
9  case 2:      *to = *from++;
10 case 1:      *to = *from++;
11         } while (--n > 0);
12 }
```

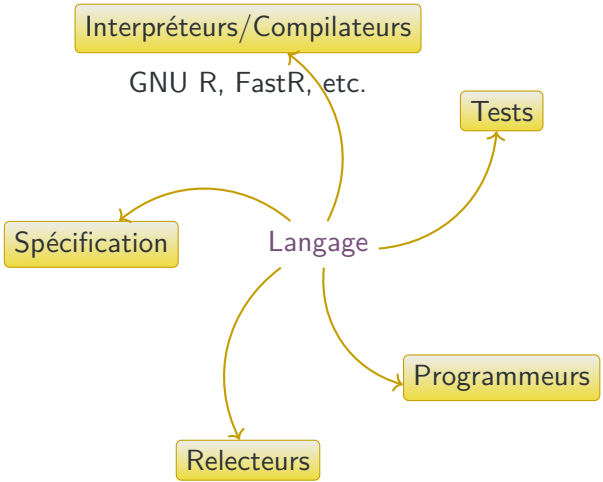
Quelle est la définition d'un langage ?

Cela dépend à qui on demande.



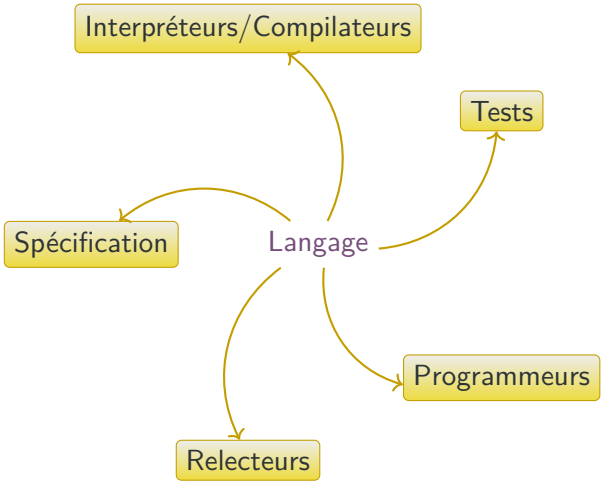
Quelle est la définition d'un langage ?

Cela dépend à qui on demande.



Quelle est la définition d'un langage ?

Cela dépend à qui on demande.

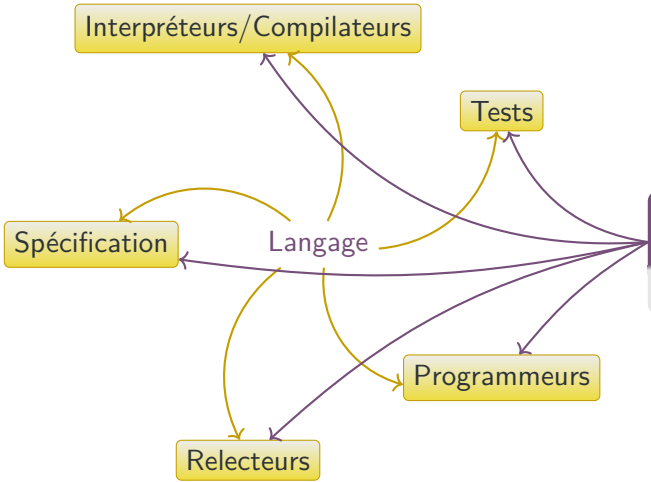


Qu'est ce qui est considéré comme un bug par la communauté ?

Cela définit les limites du langage.

Quelle est la définition d'un langage ?

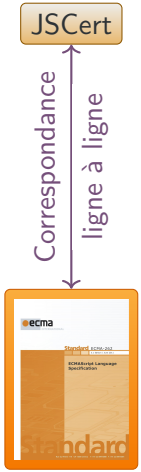
Cela dépend à qui on demande.



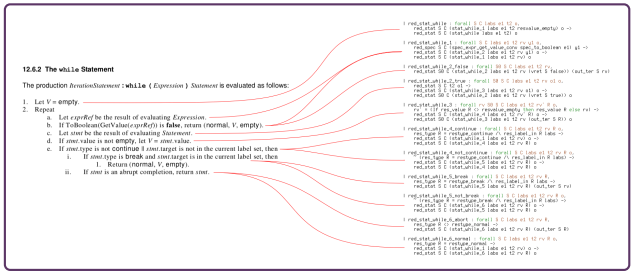
Qu'est ce qui est considéré comme un bug par la communauté ?

Cela définit les limites du langage.

JSCert : faire confiance à JavaScript



Spécification

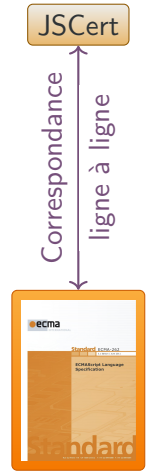


12.6.2 The while Statement

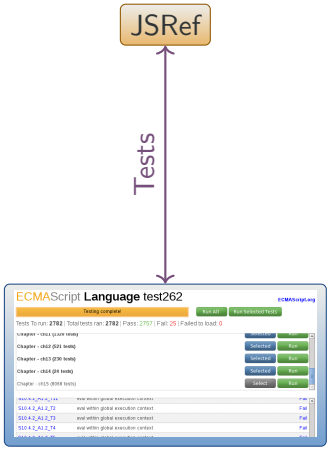
The production *IterationStatement* : **while** (*Expression*) *Statement* is evaluated as follows:

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*.

JSCert : faire confiance à JavaScript

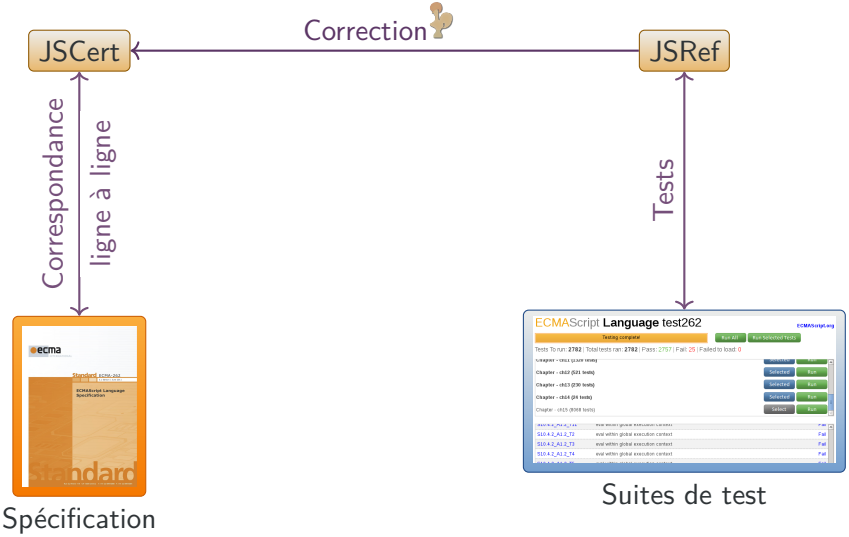


Spécification

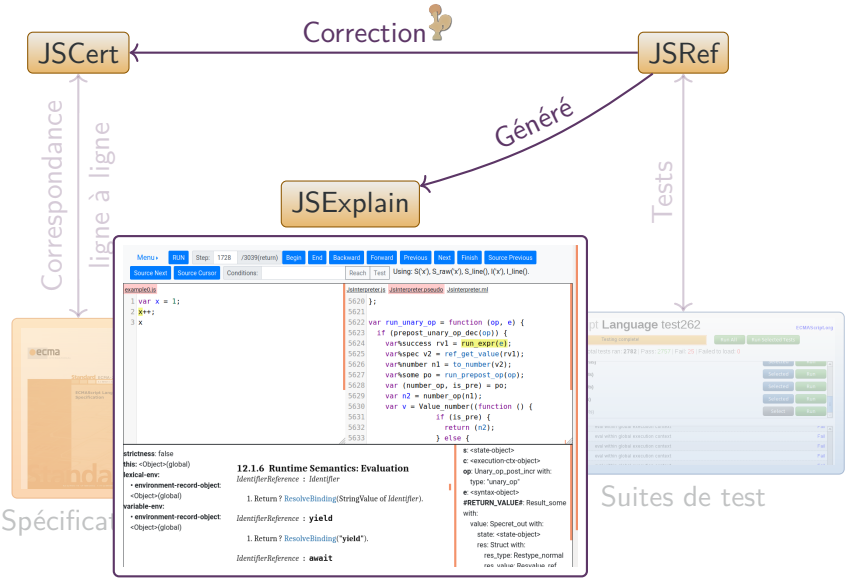


Suites de test

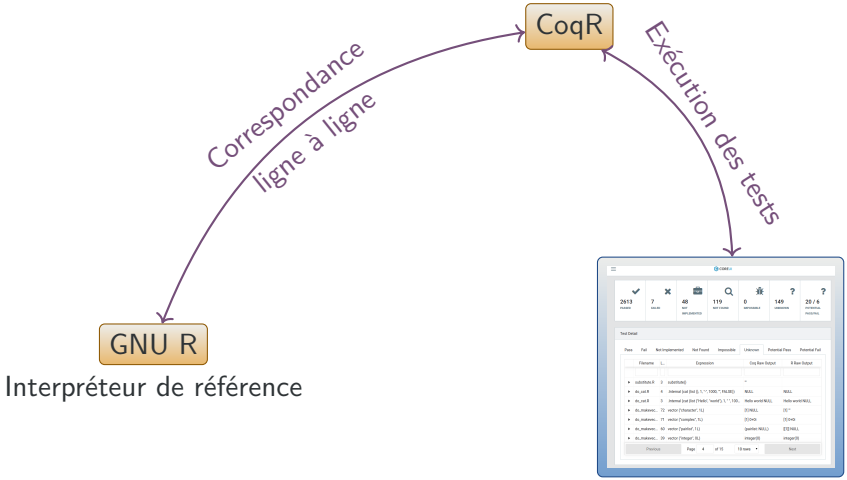
JSCert : faire confiance à JavaScript



JSCert : faire confiance à JavaScript



CoqR : faire confiance à R



Correspondance ligne à ligne

ECMAScript

"s1 ; s2" is evaluated as follows.

1. Let o_1 be the result of evaluating s1.
2. If o_1 is an aborting result, return o_1.
3. Let o_2 be the result of evaluating s2.
4. If an exception V was thrown, return $(\texttt{throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

JSCert

$$\textsc{seq-1}(s_1, s_2)\quad \frac{S, C, s_1 \Downarrow o_1 \qquad o_1, \mathit{seq}_1\ s_2 \Downarrow o}{S, C, \mathit{seq}\ s_1\ s_2 \Downarrow o}$$

$$\textsc{seq-2}(s_2)\quad \frac{}{o_1, \mathit{seq}_1\ s_2 \Downarrow o_1}\ \textbf{abort}\ o_1$$

$$\textsc{seq-3}(s_2)\quad \frac{o_1, s_2 \Downarrow o_2 \qquad o_1, o_2, \mathit{seq}_2 \Downarrow o}{o_1, \mathit{seq}_1\ s_2 \Downarrow o}\ \neg\textbf{abort}\ o_1$$

Correspondance ligne à ligne

ECMAScript

"s1 ; s2" is evaluated as follows.

1. Let o_1 be the result of evaluating s1.
2. If o_1 is an aborting result, return o_1.
3. Let o_2 be the result of evaluating s2.
4. If an exception V was thrown, return $(\texttt{throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

JSCert

$$\text{SEQ-1}(s_1, s_2)\quad \frac{S, C, s_1 \Downarrow o_1 \qquad o_1, \mathit{seq}_1\ s_2 \Downarrow o}{S, C, \mathit{seq}\ s_1\ s_2 \Downarrow o}$$

$$\text{SEQ-2}(s_2)\quad \frac{}{o_1, \mathit{seq}_1\ s_2 \Downarrow o_1}\ \textbf{abort}\ o_1$$

$$\text{SEQ-3}(s_2)\quad \frac{o_1, s_2 \Downarrow o_2 \qquad o_1, o_2, \mathit{seq}_2 \Downarrow o}{o_1, \mathit{seq}_1\ s_2 \Downarrow o}\ \neg\textbf{abort}\ o_1$$

Correspondance ligne à ligne

ECMAScript

"s1 ; s2" is evaluated as follows.

1. Let o_1 be the result of evaluating s1.
2. If o_1 is an aborting result, return o_1.
3. Let o_2 be the result of evaluating s2.
4. If an exception V was thrown, return $(\texttt{throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

JSCert

$$\text{SEQ-1}(s_1, s_2)\quad \frac{S, C, s_1 \Downarrow o_1 \qquad o_1, \mathit{seq}_1\ s_2 \Downarrow o}{S, C, \mathit{seq}\ s_1\ s_2 \Downarrow o}$$

$$\text{SEQ-2}(s_2)\quad \frac{}{o_1, \mathit{seq}_1\ s_2 \Downarrow o_1}\ \textbf{abort}\ o_1$$

$$\text{SEQ-3}(s_2)\quad \frac{o_1, s_2 \Downarrow o_2 \qquad o_1, o_2, \mathit{seq}_2 \Downarrow o}{o_1, \mathit{seq}_1\ s_2 \Downarrow o}\ \neg\textbf{abort}\ o_1$$

GNU R

```
1  SEXP do_attr
2      (SEXP call, SEXP op, SEXP args, SEXP env){
3    SEXP argList, car, ans;
4    int nargs = R_length (args);
5    argList =
6      matchArgs (do_attr_formals, args, call);
7    PROTECT (argList);
8    if (nargs < 2 || nargs > 3)
9      error ("Wrong argument count.");
10   car = CAR (argList);
11   /* ... */
12   return ans;
13 }
```

CoqR

```
1  Definition do_attr globals runs
2     (call op args env : SEXP) : result SEXP :=
3   let%success nargs :=
4     R_length globals runs args in
5   let%success argList :=
6     matchArgs globals runs
7       do_attr_formals args call in
8   if nargs <? 2 || nargs >? 3 then
9     result_error "Wrong argument count."
10  else
11    read%list car, _, _ := argList in
12    (* ... *)
13    result_success ans.
```

Correspondance ligne à ligne

ECMAScript

"s1 ; s2" is evaluated as follows.

1. Let o_1 be the result of evaluating s1.
2. If o_1 is an aborting result, return o_1.
3. Let o_2 be the result of evaluating s2.
4. If an exception V was thrown, return $(\texttt{throw}, V, \mathit{empty})$.
5. If $o_2.\mathit{value}$ is empty, let $V = o_1.\mathit{value}$, otherwise let $V = o_2.\mathit{value}$.
6. Return $(o_2.\mathit{type}, V, o_2.\mathit{target})$.

JSCert

$$\text{SEQ-1}(s_1, s_2)\quad \frac{S, C, s_1 \Downarrow o_1 \qquad o_1, \mathit{seq}_1\ s_2 \Downarrow o}{S, C, \mathit{seq}\ s_1\ s_2 \Downarrow o}$$

$$\text{SEQ-2}(s_2)\quad \frac{}{o_1, \mathit{seq}_1\ s_2 \Downarrow o_1}\ \textbf{abort}\ o_1$$

$$\text{SEQ-3}(s_2)\quad \frac{o_1, s_2 \Downarrow o_2 \qquad o_1, o_2, \mathit{seq}_2 \Downarrow o}{o_1, \mathit{seq}_1\ s_2 \Downarrow o}\ \neg\textbf{abort}\ o_1$$

GNU R

```
1  SEXP do_attr
2      (SEXP call, SEXP op, SEXP args, SEXP env){
3    SEXP argList, car, ans;
4    int nargs = R_length (args);
5    argList =
6      matchArgs (do_attr_formals, args, call);
7    PROTECT (argList);
8    if (nargs < 2 || nargs > 3)
9      error ("Wrong argument count.");
10   car = CAR (argList);
11   /* ... */
12   return ans;
13 }
```

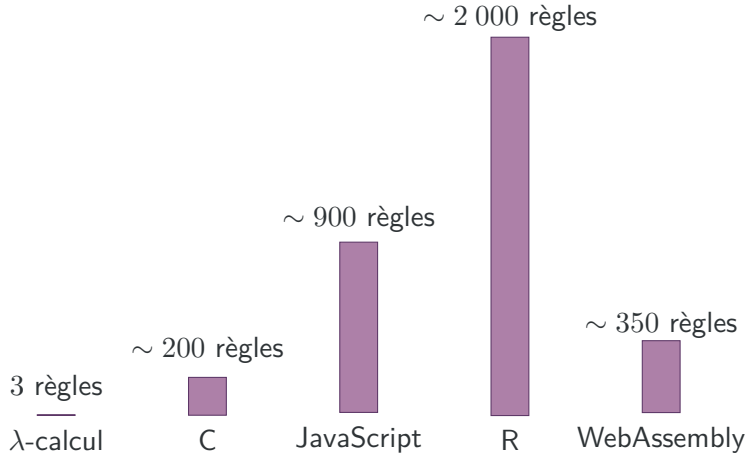
CoqR

```
1  Definition do_attr globals runs
2    (call op args env : SEXP) : result SEXP :=
3    let%success nargs :=
4      R_length globals runs args in
5    let%success argList :=
6      matchArgs globals runs
7        do_attr_formals args call in
8    if nargs <? 2 || nargs >? 3 then
9      result_error "Wrong argument count."
10   else
11     read%list car, _, _ := argList in
12     (* ... *)
13     result_success ans.
```

Preuve dans JSCert

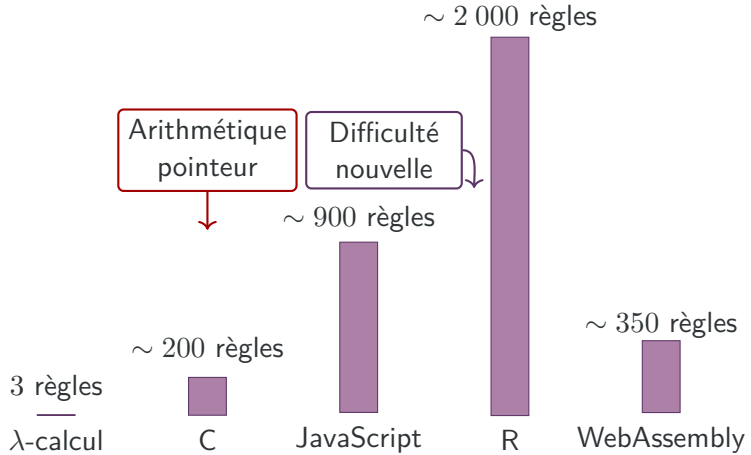
```
1  Lemma one_plus_one_exec : forall S C,
2    red_expr S C (expr_binary_op one binary_op_add one) (out_ter S (prim_number two)).
3  Proof.
4    intros.
5    eapply red_expr_binary_op.
6     constructor.
7     eapply red_spec_expr_get_value.
8      eapply red_expr_literal. reflexivity.
9     eapply red_spec_expr_get_value_1.
10    eapply red_spec_ref_get_value_value.
11   eapply red_expr_binary_op_1.
12    eapply red_spec_expr_get_value.
13     eapply red_expr_literal. reflexivity.
14    eapply red_spec_expr_get_value_1.
15    eapply red_spec_ref_get_value_value.
16   eapply red_expr_binary_op_2.
17   eapply red_expr_binary_op_add.
18    eapply red_spec_convert_twice.
19     eapply red_spec_to_primitive_pref_prim.
20    eapply red_spec_convert_twice_1.
21     eapply red_spec_to_primitive_pref_prim.
22    eapply red_spec_convert_twice_2.
23    eapply red_expr_binary_op_add_1_number.
24     simpl. intros [A|A]; inversion A.
25     eapply red_spec_convert_twice.
26      eapply red_spec_to_number_prim. reflexivity.
27     eapply red_spec_convert_twice_1.
28      eapply red_spec_to_number_prim. reflexivity.
29     eapply red_spec_convert_twice_2.
30    eapply red_expr_puremath_op_1. reflexivity.
31  Qed.
```

Tailles de sémantiques



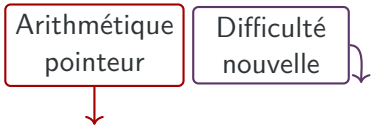
(Estimation grossière de la taille de chaque sémantique si reformulées en petit-pas.)

Tailles de sémantiques



(Estimation grossière de la taille de chaque sémantique si reformulées en petit-pas.)

Tailles de sémantiques



(Estimation grossière du nombre de fonctions basiques dans chaque sémantique.)

Squelettes

- Une syntaxe **simple** pour exprimer des sémantiques,
- Pour pouvoir en déduire des analyses statiques de programme.

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{true} \qquad \sigma, s_1 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{false} \qquad \sigma, s_2 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

Évaluation de `if e then s1 else s2` dans l'état σ

1. Soit `v` le résultat de l'évaluation de `e` dans l'état σ.
2. Si `v` est **`true`**, soit `o` le résultat de l'évaluation de `s1` dans l'état σ.
3. Si `v` est **`false`**, soit `o` le résultat de l'évaluation de `s2` dans l'état σ.
4. Retourner `o`.

Séquence

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{true} \qquad \sigma, s_1 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{false} \qquad \sigma, s_2 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

Évaluation de `if e then s1 else s2` dans l'état σ

1. Soit `v` le résultat de l'évaluation de `e` dans l'état σ.
2. Si `v` est **`true`**, soit `o` le résultat de l'évaluation de `s1` dans l'état σ.
3. Si `v` est **`false`**, soit `o` le résultat de l'évaluation de `s2` dans l'état σ.
4. Retourner `o`.

Séquence

Récursion

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{true} \qquad \sigma, s_1 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{false} \qquad \sigma, s_2 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

Évaluation de `if e then s1 else s2` dans l'état σ

1. Soit `v` le résultat de l'évaluation de `e` dans l'état σ. Récursion
2. Si `v` est **`true`**, soit `o` le résultat de l'évaluation de `s1` dans l'état σ.
3. Si `v` est **`false`**, soit `o` le résultat de l'évaluation de `s2` dans l'état σ.
4. Retourner `o`.

Les Ingrédients d'une sémantique

Branchement

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{true} \qquad \sigma, s_1 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o} \qquad \frac{\sigma, e \Downarrow v \qquad v = \texttt{false} \qquad \sigma, s_2 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

Évaluation de `if e then s1 else s2` dans l'état σ

1. Soit `v` le résultat de l'évaluation de `e` dans l'état σ.
2. Si `v` est **`true`**, soit `o` le résultat de l'évaluation de `s1` dans l'état σ.
3. Si `v` est **`false`**, soit `o` le résultat de l'évaluation de `s2` dans l'état σ.
4. Retourner `o`.

Branchement

Atome

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{true} \qquad \sigma, s_1 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

$$\frac{\sigma, e \Downarrow v \qquad v = \texttt{false} \qquad \sigma, s_2 \Downarrow o}{\sigma, \text{if } e \text{ then } s_1 \text{ else } s_2 \Downarrow o}$$

Évaluation de `if e then s1 else s2` dans l'état σ

1. Soit `v` le résultat de l'évaluation de `e` dans l'état σ.
2. Si `v` est **`true`**, soit `o` le résultat de l'évaluation de `s1` dans l'état σ.
3. Si `v` est **`false`**, soit `o` le résultat de l'évaluation de `s2` dans l'état σ.
4. Retourner `o`.

Atome

$$[\sigma, e_1 + e_2] \coloneqq \begin{bmatrix} \mathit{let}\ v_1 = \left[\sigma, e_1\right]\ \mathit{in} \\ \mathit{let}\ v_2 = [\sigma, e_2]\ \mathit{in} \\ \mathtt{add}\,(v_1, v_2) \end{bmatrix}$$

$$[\sigma, \mathit{if}\, e\, s_1\, s_2] \coloneqq \begin{bmatrix} \mathit{let}\ v = [\sigma, e]\ \mathit{in} \\ \begin{pmatrix} \mathit{let}\ \_ = \mathtt{isTrue}\,(v)\ \mathit{in}\ [\sigma, s_1] \\ \mathit{let}\ \_ = \mathtt{isFalse}\,(v)\ \mathit{in}\ [\sigma, s_2] \end{pmatrix} \end{bmatrix}$$

Récursion

$$[\sigma, e_1 + e_2] := \begin{bmatrix} \mathit{let}\ v_1 = \left[\sigma, e_1\right]\ \mathit{in} \\ \mathit{let}\ v_2 = [\sigma, e_2]\ \mathit{in} \\ \texttt{add}\,(v_1, v_2) \end{bmatrix}$$

$$[\sigma, \mathit{if}\, e\, s_1\, s_2] := \begin{bmatrix} \mathit{let}\ v = [\sigma, e]\ \mathit{in} \\ \begin{pmatrix} \mathit{let}\ \_ = \texttt{isTrue}\,(v)\ \mathit{in}\ [\sigma, s_1] \\ \mathit{let}\ \_ = \texttt{isFalse}\,(v)\ \mathit{in}\ [\sigma, s_2] \end{pmatrix} \end{bmatrix}$$

Récursion

$$[\sigma, e_1 + e_2] \coloneqq \begin{bmatrix} \mathit{let}\ v_1 = \left[\sigma, e_1\right]\ \mathit{in} \\ \mathit{let}\ v_2 = [\sigma, e_2]\ \mathit{in} \\ \mathtt{add}\,(v_1, v_2) \end{bmatrix}$$

Séquence

$$[\sigma, \mathit{if}\, e\, s_1\, s_2] \coloneqq \begin{bmatrix} \mathit{let}\ v = [\sigma, e]\ \mathit{in} \\ \left(\begin{matrix} \mathit{let}\ \_ = \mathtt{isTrue}\,(v)\ \mathit{in}\ [\sigma, s_1] \\ \mathit{let}\ \_ = \mathtt{isFalse}\,(v)\ \mathit{in}\ [\sigma, s_2] \end{matrix} \right) \end{bmatrix}$$

Récursion

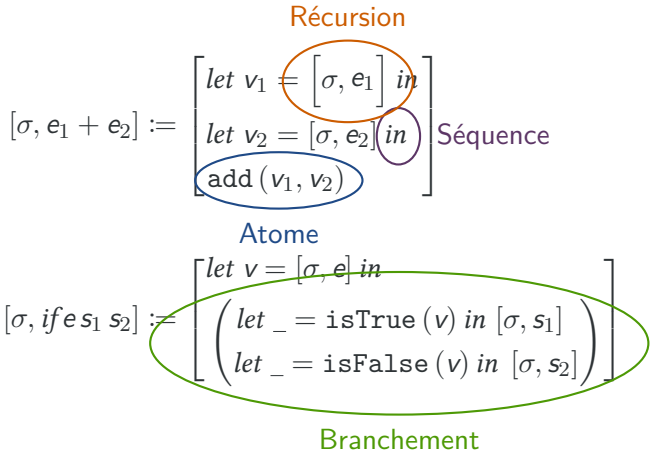
$$[\sigma, e_1 + e_2] := \left[\begin{array}{l} let\ v_1 = \left[\sigma, e_1\right]\ in \\ let\ v_2 = [\sigma, e_2]\ in \\ \texttt{add}\,(v_1, v_2) \end{array}\right]$$

Séquence

Atome

$$[\sigma, if\, e\, s_1\, s_2] := \left[\begin{array}{l} let\ v = [\sigma, e]\ in \\ \left(\begin{array}{l} let\ \_ = \texttt{isTrue}\,(v)\ in\ [\sigma, s_1] \\ let\ \_ = \texttt{isFalse}\,(v)\ in\ [\sigma, s_2] \end{array}\right) \end{array}\right]$$

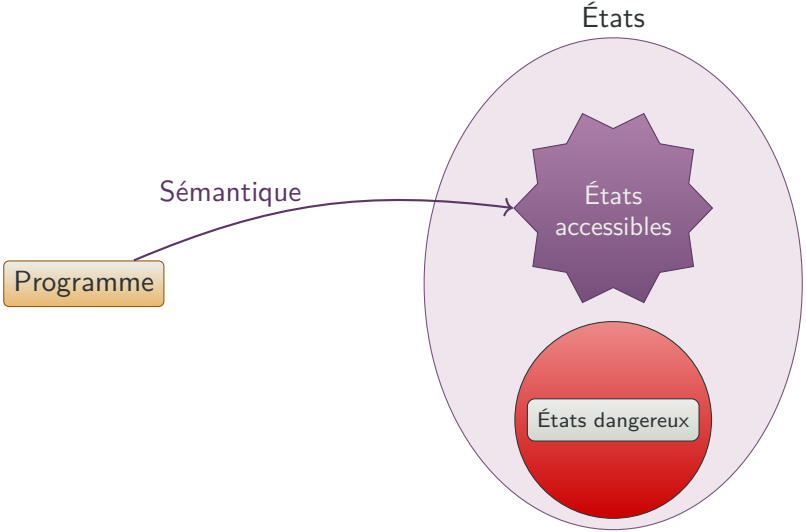
Structure d'un squelette



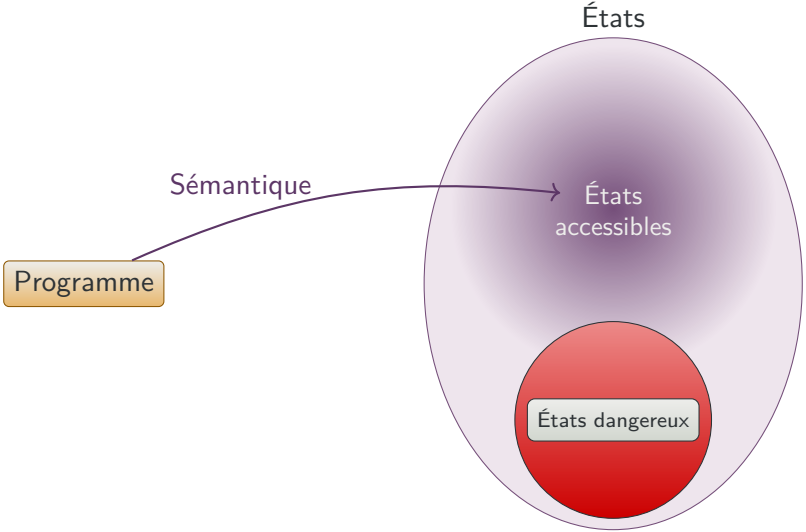
Analyses

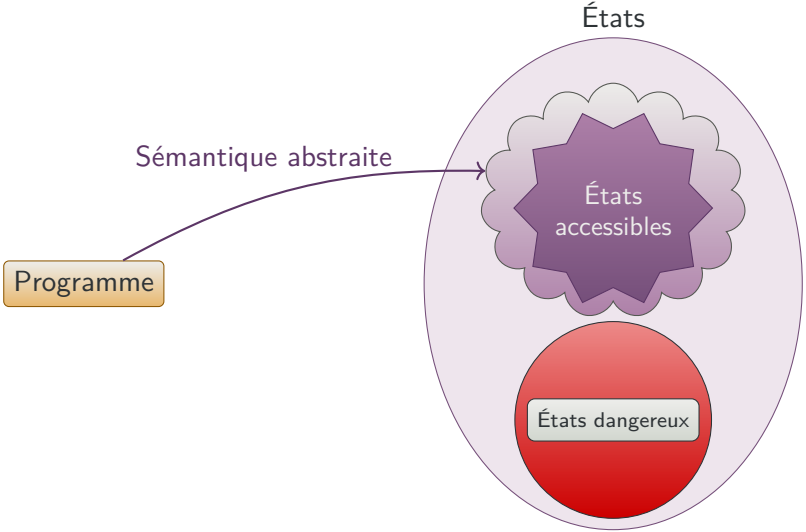
Intérêt des squelettes

Définir et montrer correcte une analyse **une fois pour toutes**,
pour l'instancier à tous les languages (via leurs squelettes).

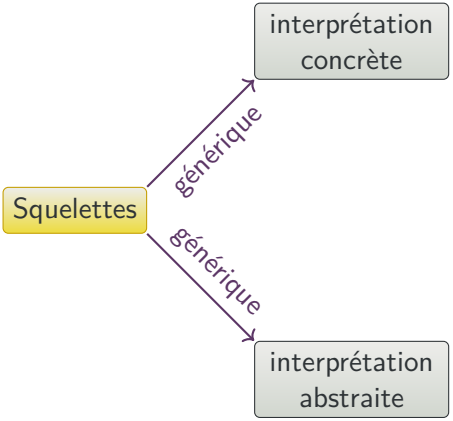

États
Sémantique
États accessibles
Programme
États dangereux

Interprétation abstraite

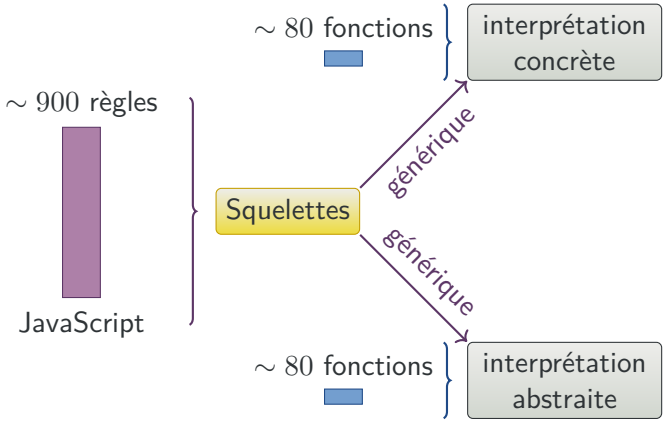



États
Sémantique abstraite
États
accessibles
Programme
États dangereux

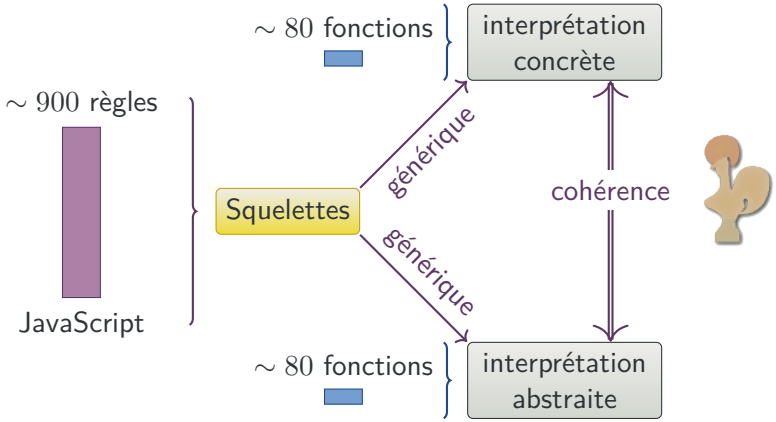
Interprétations



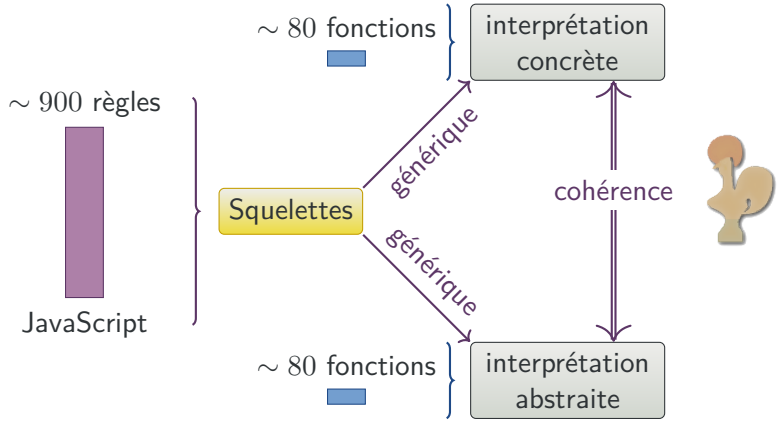
Interprétations



Interprétations



Merci pour votre attention !



martin.bodin@inria.fr

1. Des langages de programmation complexes

2. Formaliser ces langages en Coq

3. Squelettes : une syntaxe pour la sémantique

Bonus

1. Définition formelle des squelettes,
2. Interprétation concrète,
3. Interprétation abstraite,
4. Interprétation de coûts,
5. Cohérences d'interprétations,
6. JSkel.

Les squelettes, formellement

$$\textsc{Terms} \quad t ::= b \mid x_t \mid c(t_1..t_n)$$

$$\textsc{Skeleton} \quad ::= \textsc{Name}(c(x_{t_1}..x_{t_n})) := S$$

$$\textsc{Skeleton Body} \quad S ::= let\ x = S\ in\ S' \mid F(x_1..x_n) \mid [x, t] \mid \begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix}$$

- Un crochet $[x, t]$ est un appel récursif du terme t dans l'état x.
- Les filtres sont à la fois des calculs atomiques et des prédicats.

Interprétation concrète

$E : variable \to value$ $\qquad$ $[\![S]\!]\ (E) : \mathcal{P}\,(output)$

$$[\![let\ x = S\ in\ S']\!]\ (E) =$$

$$[\![[x, t]]\!]\ (E) =$$

$$[\![F(x_1..x_n)]\!]\ (E) =$$

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]\ (E) =$$

Interprétation concrète

$$E : variable \to value \qquad [\![S]\!]\ (E) : \mathcal{P}(output)$$

$$[\![let\ x = S\ in\ S']\!]\ (E) = \bigcup_{v \in [\![S]\!]\ (E)} [\![S']\!]_T (E + x \mapsto v)$$

$$[\![[x, t]]\!]\ (E) =$$

$$[\![F(x_1..x_n)]\!]\ (E) = [\![F]\!]\,(E[x_1]..E[x_n])$$

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]\ (E) = \bigcup_{0 < i \leqslant n} [\![S_i]\!]_T (E)$$

Interprétation concrète

$$E : variable \to value \qquad [\![S]\!]_T(E) : \mathcal{P}(output)$$

$$[\![let\ x = S\ in\ S']\!]_T(E) = \bigcup_{v \in [\![S]\!]_T(E)} [\![S']\!]_T(E + x \mapsto v)$$

$$[\![[x,t]]\!]_T(E) = \{v \mid (E[x], t, v) \in T\}$$

$$[\![F(x_1..x_n)]\!]_T(E) = [\![F]\!](E[x_1]..E[x_n])$$

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]_T(E) = \bigcup_{0<i\leqslant n} [\![S_i]\!]_T(E)$$

Interprétation concrète

$E : variable \to value$ $\qquad\qquad$ $[\![S]\!]_T(E) : \mathcal{P}(output)$

$$[\![let\ x = S\ in\ S']\!]_T(E) = \bigcup_{v \in [\![S]\!]_T(E)} [\![S']\!]_T(E + x \mapsto v)$$

$$[\![[x, t]]\!]_T(E) = \{v \mid (E[x], t, v) \in T\}$$

$$[\![F(x_1..x_n)]\!]_T(E) = [\![F]\!](E[x_1]..E[x_n])$$

Dépendent du langage

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]_T (E) = \bigcup_{0 < i \leqslant n} [\![S_i]\!]_T(E)$$

Interprétation abstraite

$$E^\sharp : variable \to value^\sharp \qquad [\![S]\!]_T\left(E^\sharp\right) : \mathcal{P}\left(output^\sharp\right)$$

$$[\![let\ x = S\ in\ S']\!]_T\left(E^\sharp\right) = \bigcup_{v^\sharp \in [\![S]\!]_T\left(E^\sharp\right)} [\![S']\!]_T\left(E^\sharp + x \mapsto v^\sharp\right)$$

$$[\![[x,t]]\!]_T\left(E^\sharp\right) = \left\{ v^\sharp \;\middle|\; \left(E^\sharp[x], t, v^\sharp\right) \in T \right\}$$

$$[\![F(x_1..x_n)]\!]_T\left(E^\sharp\right) = [\![F]\!]^\sharp\left(E^\sharp[x_1]..E^\sharp[x_n]\right)$$

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]_T\left(E^\sharp\right) = \bigcap_{0<i\leqslant n} [\![S_i]\!]_T\left(E^\sharp\right)$$

Interprétation abstraite

$$E^\sharp : variable \to value^\sharp \qquad [\![S]\!]_T\left(E^\sharp\right) : \mathcal{P}\left(output^\sharp\right)$$

$$[\![let\ x = S\ in\ S']\!]_T\left(E^\sharp\right) = \bigcup_{v^\sharp \in [\![S]\!]_T\left(E^\sharp\right)} [\![S']\!]_T\left(E^\sharp + x \mapsto v^\sharp\right)$$

$$[\![[x,t]]\!]_T\left(E^\sharp\right) = \left\{ v^\sharp \;\middle|\; \left(E^\sharp[x], t, v^\sharp\right) \in T \right\}$$

$$[\![F(x_1..x_n)]\!]_T\left(E^\sharp\right) = [\![F]\!]^\sharp\left(E^\sharp[x_1]..E^\sharp[x_n]\right)$$

Dépendent du langage

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]_T\left(E^\sharp\right) = \bigcap_{0<i\leqslant n} [\![S_i]\!]_T\left(E^\sharp\right)$$

Interprétation de coûts

$$[\![S]\!]_T : cost$$

$$[\![let\ x = S\ in\ S']\!]_T =$$

$$[\![[x, t]]\!]_T =$$

$$[\![F(x_1..x_n)]\!]_T = [\![F]\!]$$

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]_T =$$

Interprétation de coûts

$$[\![S]\!]_T : cost$$

$$[\![let\ x = S\ in\ S']\!]_T = \max_{\substack{c \in [\![S]\!]_T \\ c' \in [\![S']\!]_T}} c + c'$$

$$[\![[x, t]]\!]_T = \max \{c \mid (t, c) \in T\}$$

$$[\![F(x_1..x_n)]\!]_T = [\![F]\!]$$

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]_T = \max_{0 < i \leqslant n} [\![S_i]\!]_T$$

Interprétation de coûts

$$[\![S]\!]_T : cost$$

$$[\![let\ x = S\ in\ S']\!]_T = \max_{\substack{c \in [\![S]\!]_T \\ c' \in [\![S']\!]_T}} c + c'$$

$$[\![[x,t]]\!]_T = \max\{c \mid (t,c) \in T\}$$

$$[\![F(x_1..x_n)]\!]_T = [\![F]\!]$$

Dépend des entrées de F

$\Rightarrow$ composer avec les interprétations concrètes et abstraites

$$\left[\!\!\left[\begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix} \right]\!\!\right]_T = \max_{0<i\leqslant n} [\![S_i]\!]_T$$

Correction de la sémantique abstraite

Théorème

$$\forall \left(\sigma^\sharp, t, v^\sharp\right) \in \Downarrow^\sharp. \ \forall (\sigma, t, v) \in \Downarrow. \ \text{Si } \sigma \in \gamma\left(\sigma^\sharp\right) \text{ alors } v \in \gamma\left(v^\sharp\right)$$

Preuve

Grâce aux cohérences d'interprétation.

Cohérence

- un prédicat sur les entrées des deux interprétations $OKst\left((E, T), \left(E^\sharp, T^\sharp\right)\right)$
- un prédicat sur les sorties des deux interprétations $OKout\left(v, v^\sharp\right)$

Cohérence

Si la cohérence des deux prédicats est préservée par l'action des quatres constructeurs des squelettes, alors elle est préservée de manière globale.

Cohérence

Si la cohérence des deux prédicats est préservée par l'action des quatres constructeurs des squelettes, alors elle est préservée de manière globale.

Cohérence entre les interprétations concrète et abstraite

$$
\begin{aligned}
OKst\left((E, T), \left(E^\sharp, T^\sharp\right)\right) ::= \; & dom\,(E) = dom\left(E^\sharp\right) \\
& \wedge \forall x.\ E[x] \in \gamma\left(E^\sharp[x]\right) \\
& \wedge \forall (v_1, t, v_2) \in T, (v_1^\sharp, t, v_2^\sharp) \in T^\sharp. \\
& \qquad v_1 \in \gamma\left(v_1^\sharp\right) \Rightarrow v_2 \in \gamma\left(v_2^\sharp\right)
\end{aligned}
$$

Cohérence

Si la cohérence des deux prédicats est préservée par l'action des quatres constructeurs des squelettes, alors elle est préservée de manière globale.

Cohérence entre les interprétations concrète et abstraite

$$
\begin{aligned}
OKst\left((E,T),\left(E^\sharp,T^\sharp\right)\right) ::= \; & dom\,(E) = dom\left(E^\sharp\right) \\
& \wedge \forall x.\; E[x] \in \gamma\left(E^\sharp[x]\right) \\
& \wedge \forall (v_1,t,v_2) \in T, (v_1^\sharp,t,v_2^\sharp) \in T^\sharp. \\
& \qquad v_1 \in \gamma\left(v_1^\sharp\right) \Rightarrow v_2 \in \gamma\left(v_2^\sharp\right)
\end{aligned}
$$

Théorème

Si les atomes sont cohérents, alors la sémantique abstraite est correcte.

JSkel : Un squelette pour JavaScript

```
1  val ifStatement_Evaluation (ifSTMT : ifStatement) : st<out<maybeEmpty<value>>> =
2    let res =@r
3       branch let IfElse (expr, stmt1, stmt2) = ifSTMT in
4              let exprRef =@s expression_Evaluation(expr) in
5              let exprValue =? getValue(exprRef) in let exprValue = toBoolean(exprValue) in let stmtCompletion =$ Any in
6              branch let T = exprValue in
7                     let v =@cr statement_Evaluation(stmt1) in (stmtCompletion, CR_ME_Val v);^ cont<maybeEmpty<value>> ()
8              or     let F = exprValue in
9                     let v =@cr statement_Evaluation(stmt2) in (stmtCompletion, CR_ME_Val v);^ cont<maybeEmpty<value>> ()
10             end;@ let CR_ME_Val stmtCompletion =* stmtCompletion in
11             let r =@cr updateEmpty(stmtCompletion, (NotEmpty<value> Undefined)) in completion r
12      or     let If (expr, stmt) = ifSTMT in
13             let exprRef =@s expression_Evaluation(expr) in
14             let exprValue =? getValue(exprRef) in let exprValue = toBoolean(exprValue) in
15             branch let F = exprValue in
16                    ret<maybeEmpty<value>> (NotEmpty<value> Undefined)
17             or     let T = exprValue in
18                    let stmtCompletion =@cr statement_Evaluation(stmt) in
19                    let r =@cr updateEmpty(stmtCompletion, (NotEmpty<value> Undefined)) in completion r
20             end
21      end
22   in res
```

gitlab.inria.fr/skeletons/jskel

1. Définition formelle des squelettes,
2. Interprétation concrète,
3. Interprétation abstraite,
4. Interprétation de coûts,
5. Cohérences d'interprétations,
6. JSkel.

1. Des langages de programmation complexes

2. Formaliser ces langages en Coq

3. Squelettes : une syntaxe pour la sémantique