Auditions Inria CR/ISFP

Grenoble — Rhône–Alpes

Équipe d'accueil : Spades 

Martin Bodin

17 juin

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).
  Squelettes
  WebAssembly en Coq

Mon Credo

- Les langages de programmation **populaires** sont **complexes**,
- Leur complexité est une **source d'erreurs** importante,
- Il y a ainsi un besoin d'**analyse** et d'**outils** pour ces langages,
  - Détecter des débordements de mémoire, divisions par 0, etc.
  - Intégration continue.

Infer


Sapienz

- Les langages sont complexes $\Longrightarrow$ le code des outils est complexe,
  $\Longrightarrow$ **erreur dans l'outil** probable,
  $\Longrightarrow$ crise de la **confiance**.
- L'**assistant de preuve Coq**  est nécessaire pour ces outils.

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

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

```
1  (+(+!+[]+(!+[]+[])[!+[]+!+[]+!+[]]+[+!+[]]+[+[]]+[+[]]
2    +[+[]])+[])[+[]]+([][[]]+[])[+!+[]]+(!![]+[])[+!+[]]
3    +([![]]+[][[]])[+!+[]+[+[]]]+(![]+[])[+!+[]]
4   →  "Inria"
```

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

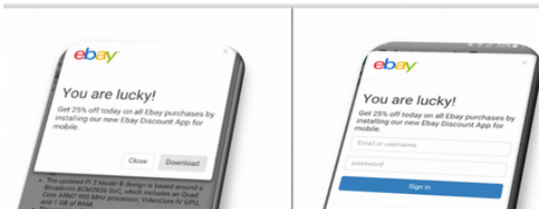
```
1  (+(+!+[]+(!+[]+
2    +[+[]])+[])[+
3    +([![]]+[][[]
4   →   "Inria"
```

Une vulnérabilité transforme eBay en site de phishing

Dominique Filippone , publié le 04 Février 2016

10 Réaction

Découverte par Check Point, une vulnérabilité du site de vente en ligne d'eBay permet à des cybercriminels d'exécuter du code Javascript malveillant à distance et de diffuser des campagnes de phishing et des malwares. Alerté par l'éditeur de solutions de sécurité depuis décembre dernier, eBay n'a, à ce jour,toujours pas corrigé son code.



SUIVRE TOUTE L'

Newsletter

Recevez notre newsl
de 50 000 professio

JE M'ABONNE

1. Les langages populaires sont complexes
2. Formaliser ces langages en Coq
3. Squelettes : une syntaxe pour la sémantique
4. Projet de recherche

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 ?

JSCert : `https://github.com/jscert/jscert`

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

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

CoqR : `https://github.com/Mbodin/CoqR`

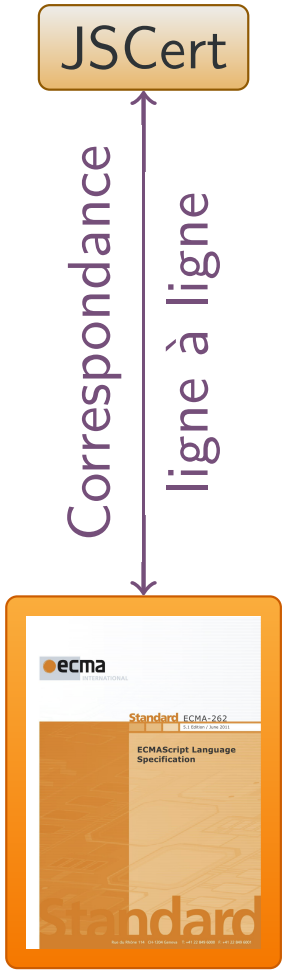
Martin Bodin. "A Coq Formalisation of a Core of R". In: *CoqPL*. 2018.

Martin Bodin, Tomás Diaz, and Éric Tanter. "A Trustworthy Mechanized Formalization of R". In: *DLS*. 2018.

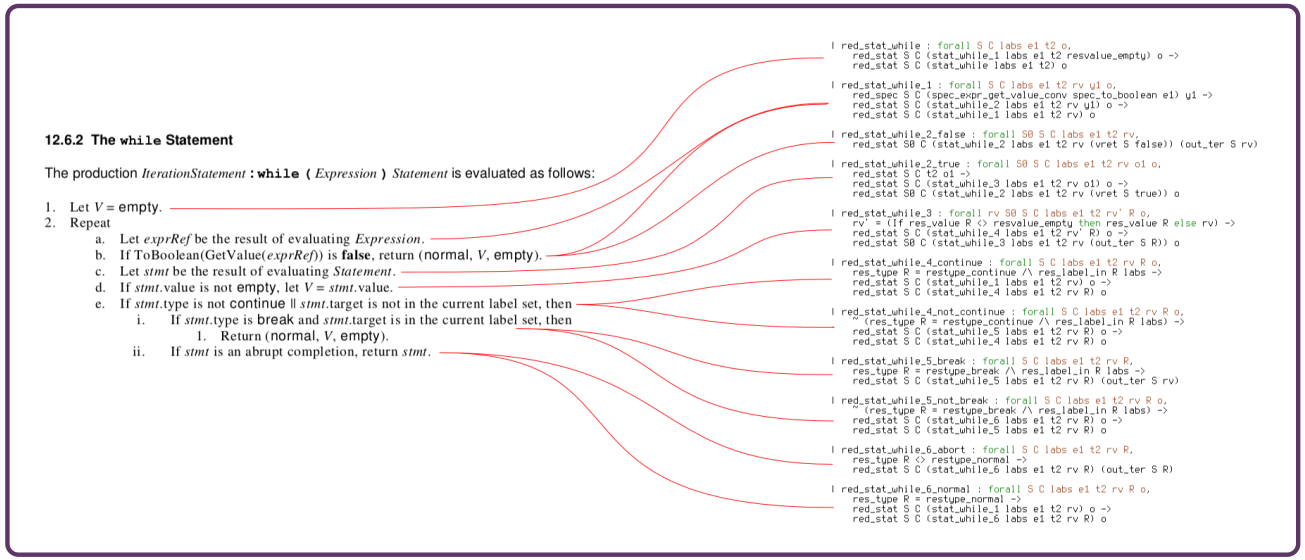
WebAssembly

Projet en cours.

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*.

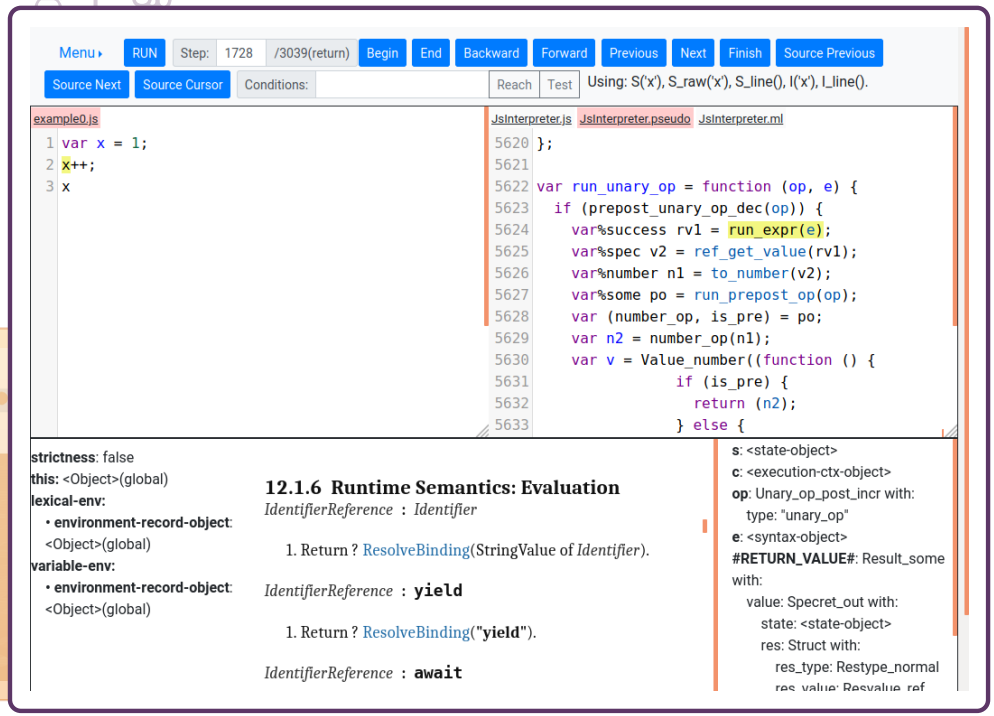
```
| red_stat_while : forall S C labs e1 t2 o,
    red_stat S C (stat_while_1 labs e1 t2 resvalue_empty) o ->
    red_stat S C (stat_while labs e1 t2) o

| red_stat_while_1 : forall S C labs e1 t2 rv 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

| 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
```

JSCert : faire confiance à JavaScript

JSCert

Tests

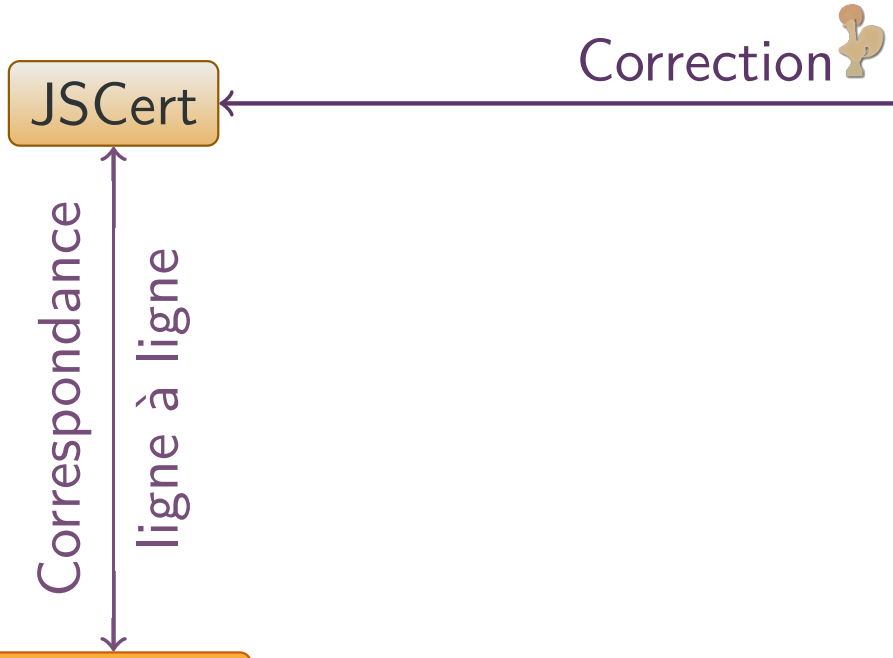
JSRef



Spécification

Suites de test

JSCert : faire confiance à JavaScript



Spécification

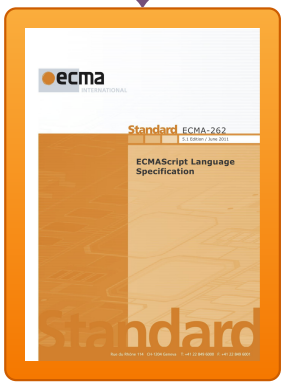
Suites de test

JSCert : faire confiance à JavaScript

Correspondance ligne à ligne

- Premier lien entre la spécification et les tests officiels,
- JSCert est digne de confiance : **référence** pour JavaScript en Coq.

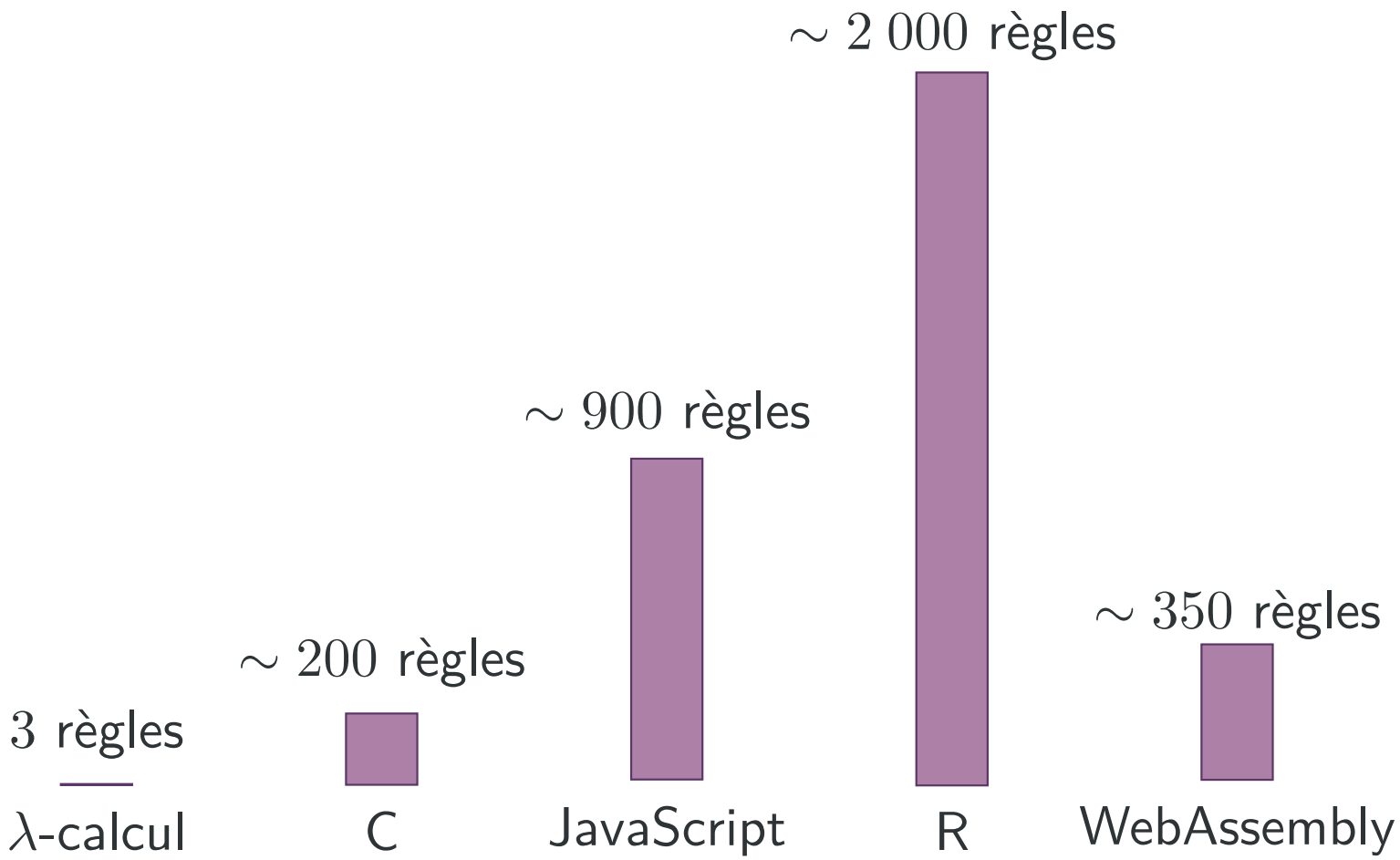
Tests



Spécification

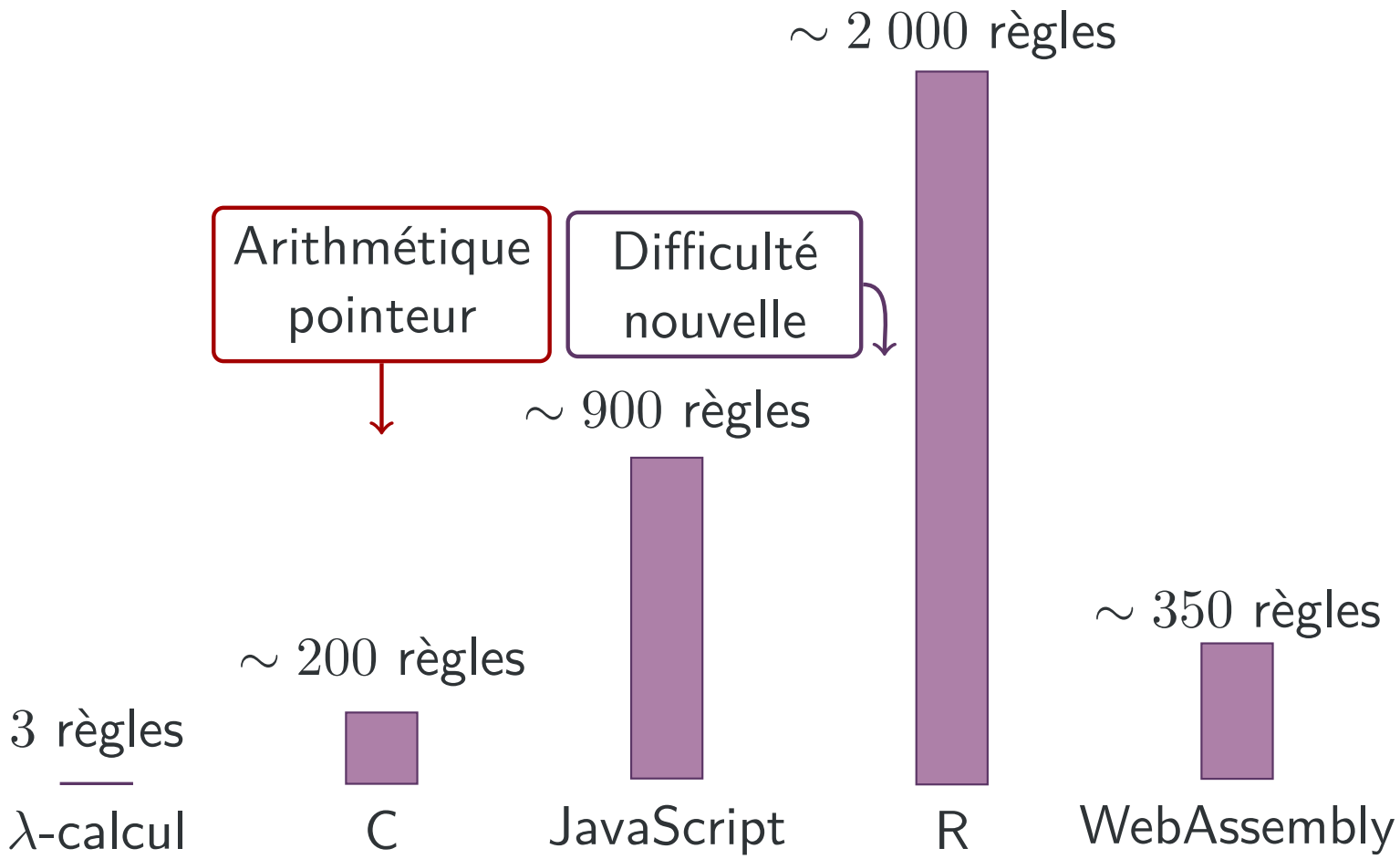
Suites de test

Tailles de sémantiques



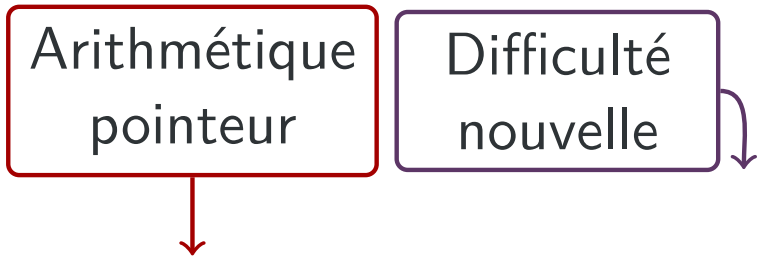
(Estimation grossière de la taille de chaque sémantique si reformulées en petit-pas.)

Tailles de sémantiques



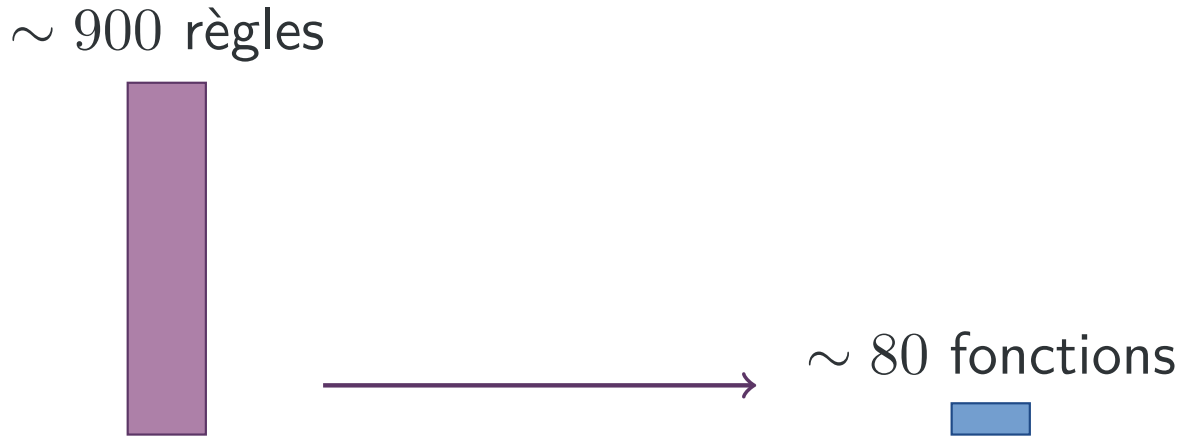
(Estimation grossière de la taille de chaque sémantique si reformulées en petit-pas.)

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.

Squelettes: `https://gitlab.inria.fr/skeletons/Coq`

Martin Bodin, Thomas Jensen, and Alan Schmitt. "Pretty-big-step-semantics-based Certified Abstract Interpretation". In: *JFLA*. 2014.

Martin Bodin, Thomas Jensen, and Alan Schmitt. "Certified Abstract Interpretation with Pretty-Big-Step Semantics". In: *CPP*. 2015.

Martin Bodin et al. "Skeletal Semantics and their Interpretations". In: *POPL* (2019).

Les Ingrédients d'une sémantique

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

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

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`.

Les Ingrédients d'une sémantique

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

Les Ingrédients d'une sémantique

Récursion

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

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. 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

Les Ingrédients d'une sémantique

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

Structure d'un squelette

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

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

Structure d'un squelette

Récursion

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

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

Structure d'un squelette

Récursion

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

Séquence

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

Structure d'un squelette

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

Récursion

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

Séquence

Atome

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

Branchement

Structure d'un squelette

Récursion

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

Séquence

Atome

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

Branchement

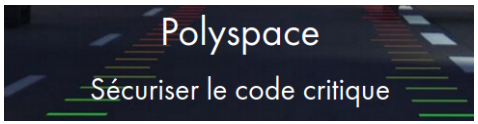
$$S ::= \quad [x, t] \quad | \quad let\ x = S\ in\ S' \quad | \quad F(x_1..x_n) \quad | \quad \begin{pmatrix} S_1 \\ .. \\ S_n \end{pmatrix}$$

Analyses

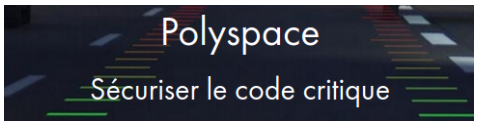
Intérêt des squelettes

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

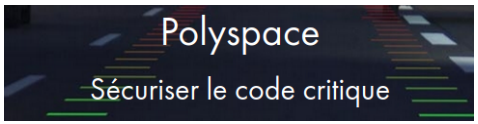
Interprétation abstraite



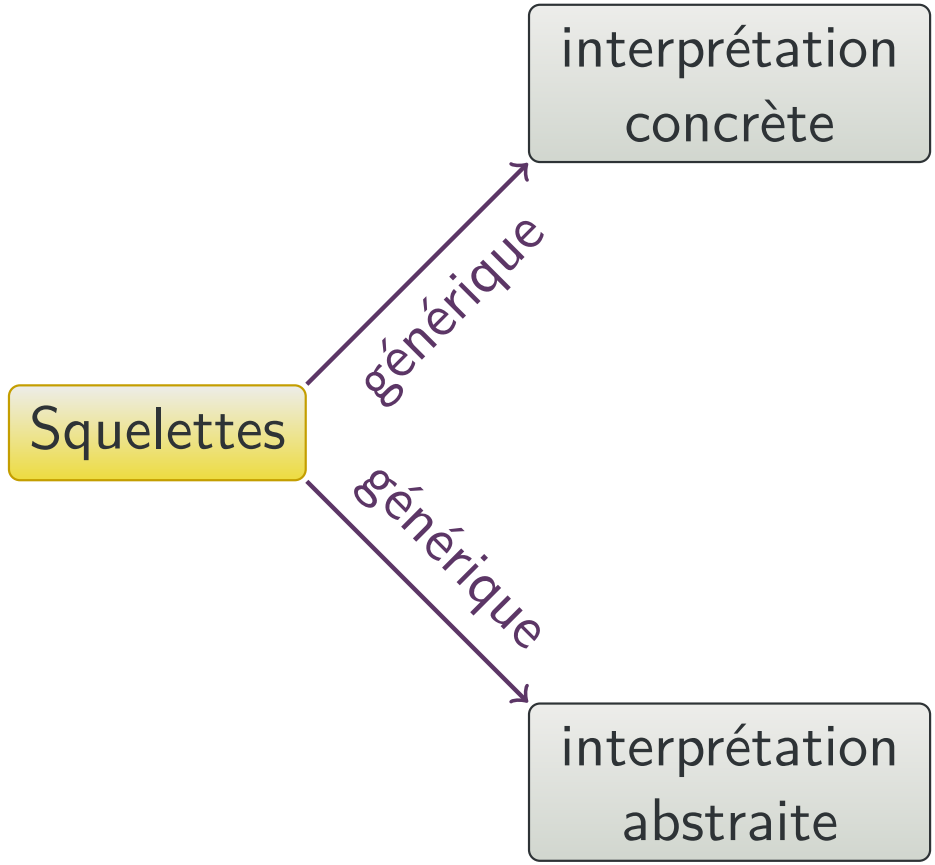
Interprétation abstraite



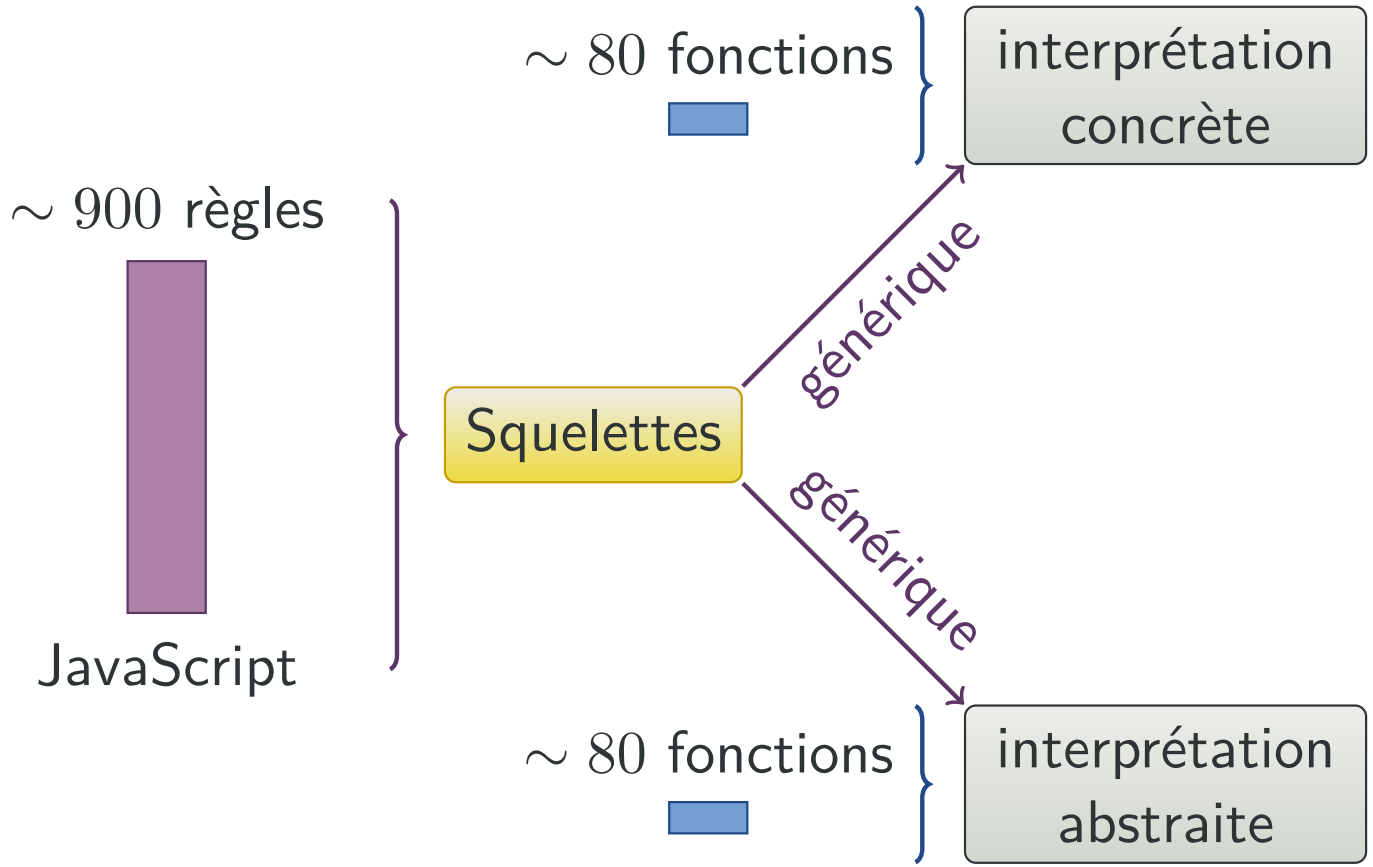
Interprétation abstraite



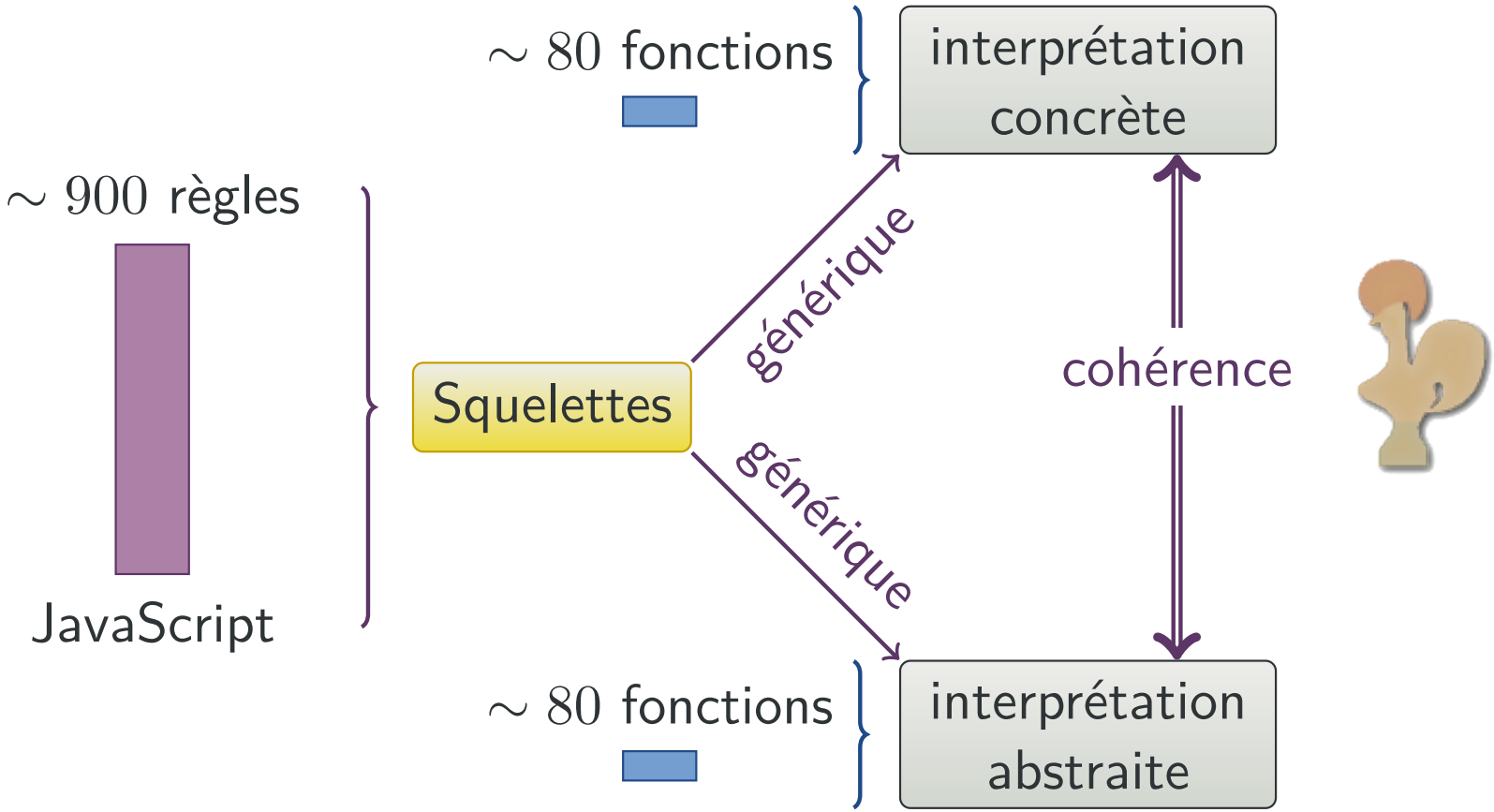
Interprétations



Interprétations



Interprétations



1. Les langages populaires sont complexes
2. Formaliser ces langages en Coq
3. Squelettes : une syntaxe pour la sémantique
4. Projet de recherche

Projet de recherche

Une infrastructure pour des analyses et des compilateurs certifiés pour des langages de programmation complexes.

- Pourquoi ?
  - Un **usage important** dans l'industrie,
  - Les sémantiques complexes sont **peu étudiées** dans leur intégralité.
- Difficultés
  - Formalisation très larges : crise de **confiance**,
  - Passage à l'échelle,
  - Des analyses clefs en main pour tous les langages.
- Comment ?
  - Les **squelettes** offrent un cadre qui **passe à l'échelle**,
  - Appliquer les squelettes à des **systèmes réels**.

Intégration dans Spades

Méthodes formelles pour les systèmes embarqués

- Modèles formels, langages, et outils ;
  - **Analyses de programme dataflow** : Pascal Fradet, Alain Girault.
  - Dataflow reconfigurable : Pascal Fradet, Alain Girault, Xavier Nicollin.
  - Hypercells : Jean-Bernard Stefani.
- Programmation temps-réel certifiée en Coq ;
  - **Analyses d'ordonnançabilité certifiées** : Pascal Fradet, Sophie Quinton.
  - **Compilation multi-critères** (temps/énergie, prédictibilité) : Alain Girault.
- Gestion des fautes et analyse causale.
  - Gregor Gössler, Jean-Bernard Stefani.

Intégration dans Spades

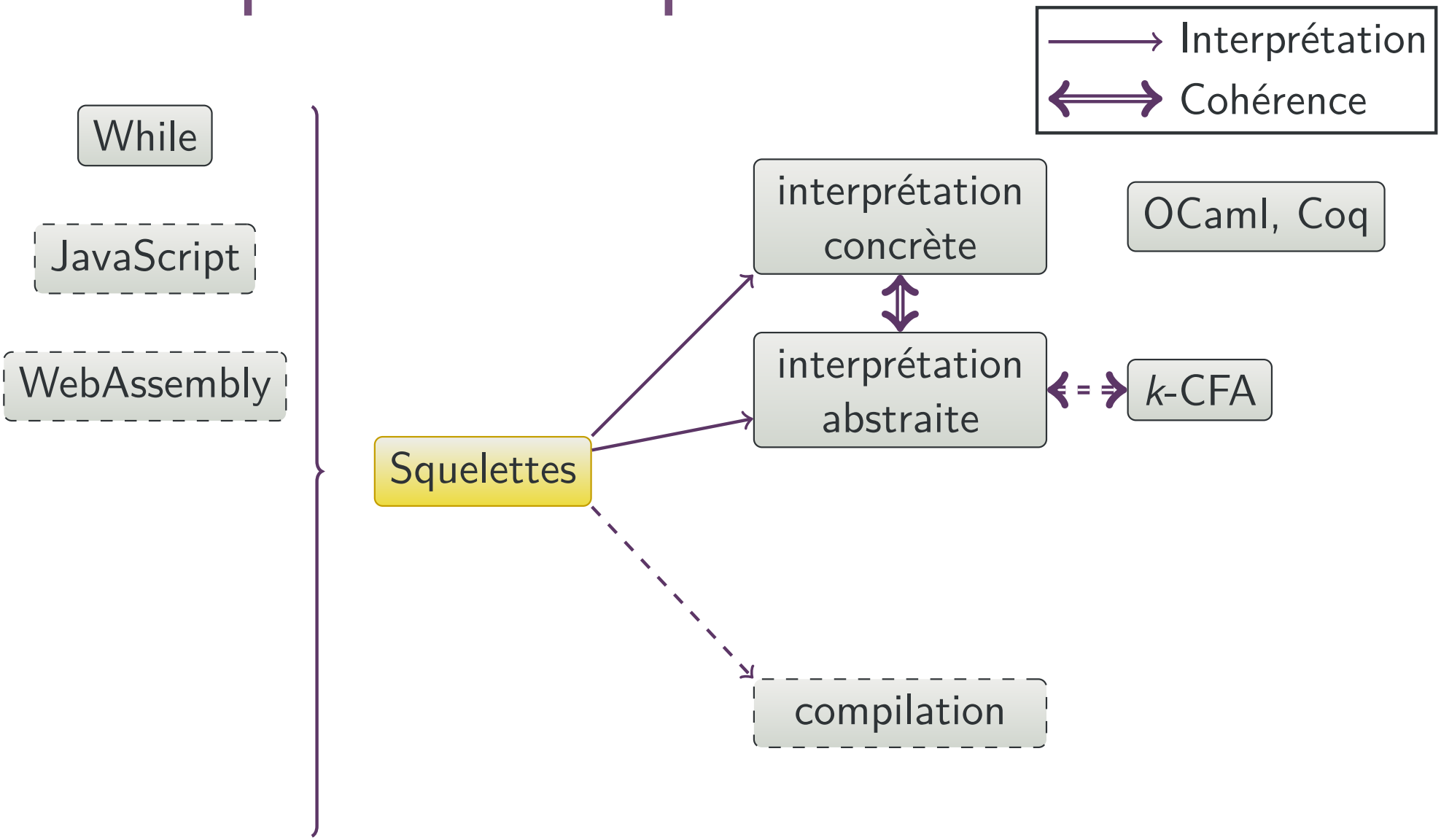
Méthodes formelles pour les systèmes embarqués

- Modèles formels, langages, et outils ;
  - **Analyses de programme dataflow** : Pascal Fradet, Alain Girault.
  - Dataflow reconfigurable : Pascal Fradet, Alain Girault, Xavier Nicollin.
  - Hypercells : Jean-Bernard Stefani.
- Programmation temps-réel certifiée en Coq ;
  - **Analyses d'ordonnançabilité certifiées** : Pascal Fradet, Sophie Quinton.
  - **Compilation multi-critères** (temps/énergie, prédictibilité) : Alain Girault.
- Gestion des fautes et analyse causale.
  - Gregor Gössler, Jean-Bernard Stefani.

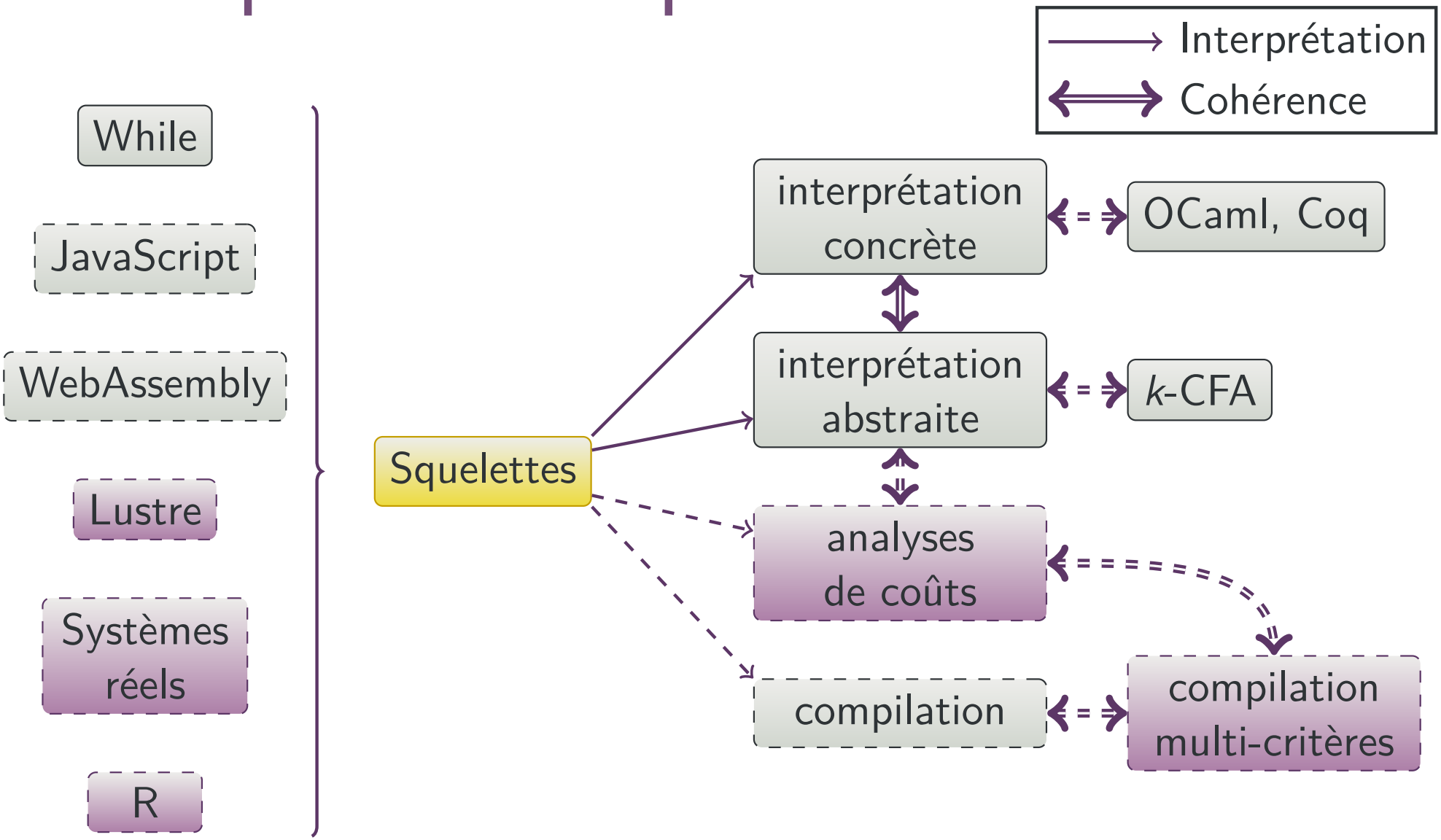
J'apporte mes compétences

- Coq,
- Analyse de programmes, interprétation abstraite, logique de séparation.
- Théorie des langages de programmation.

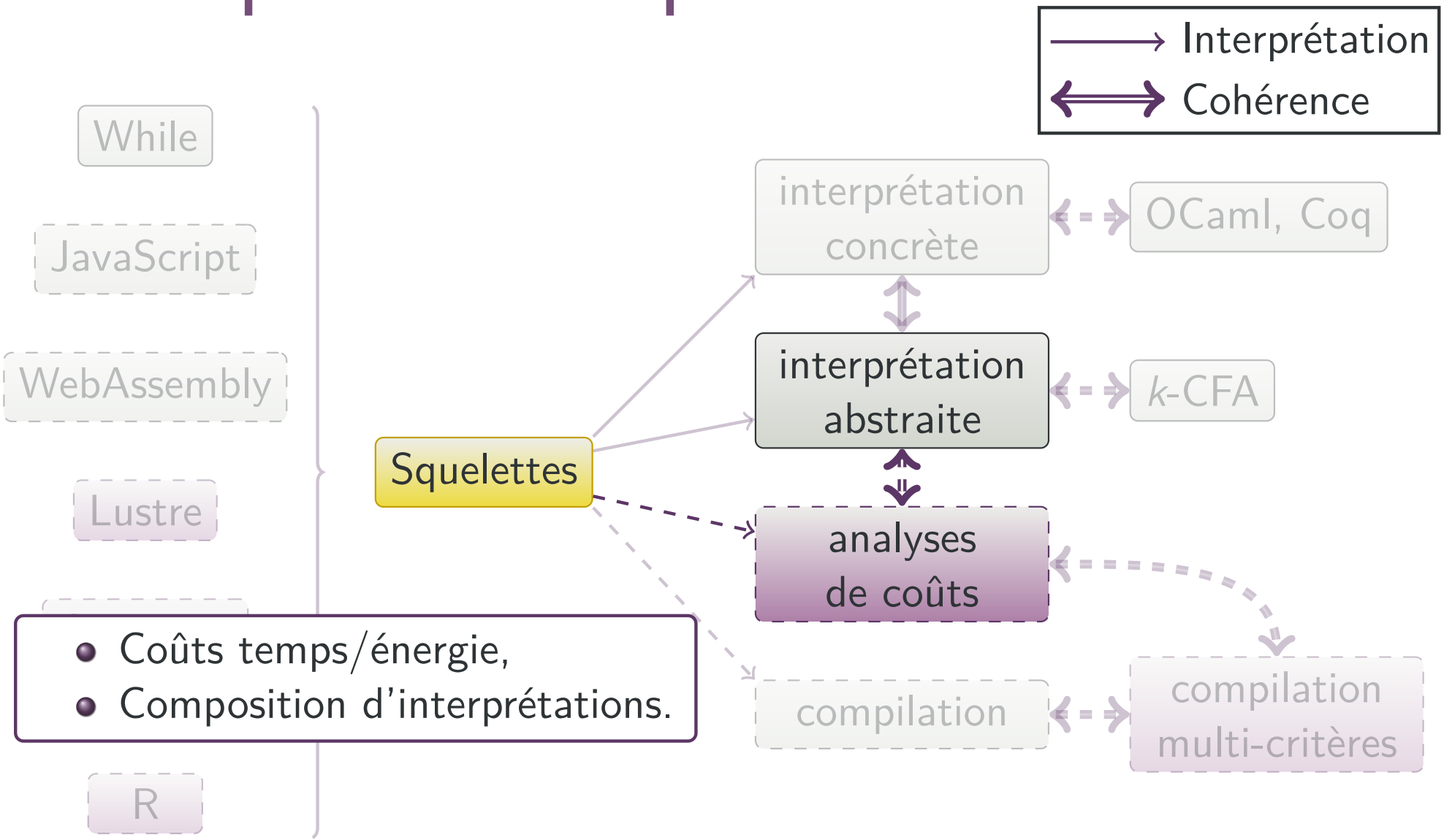
Squelettes : présent et futur



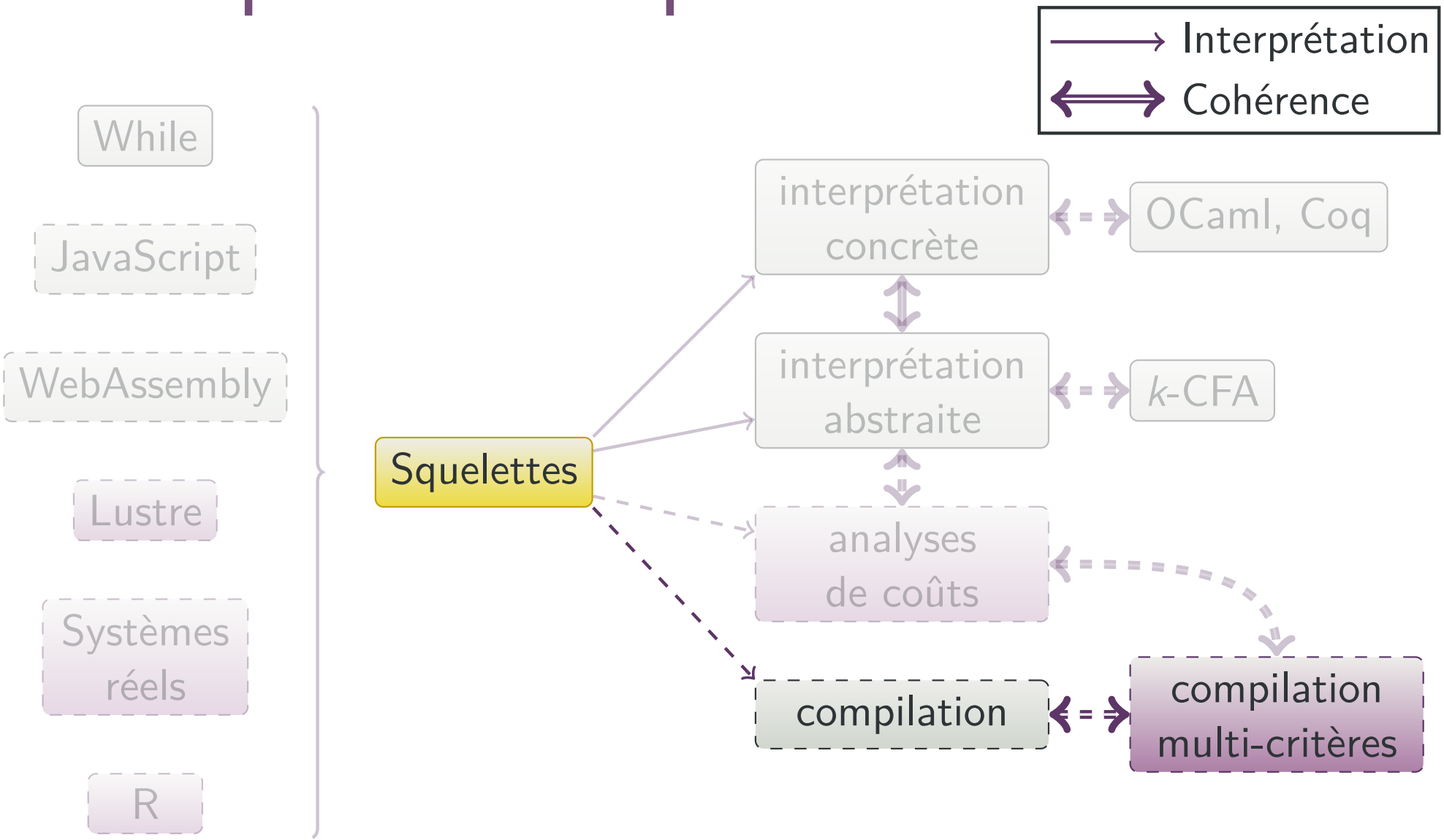
Squelettes : présent et futur



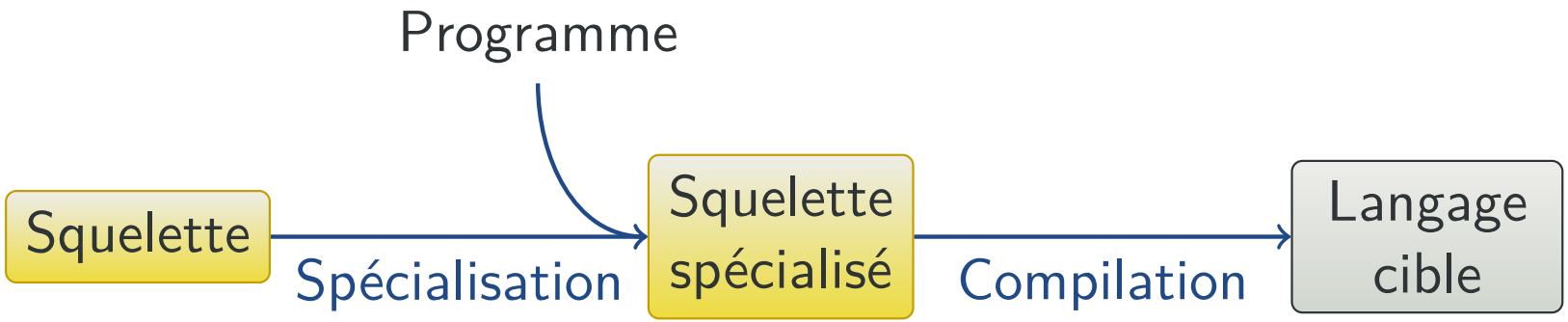
Squelettes : présent et futur



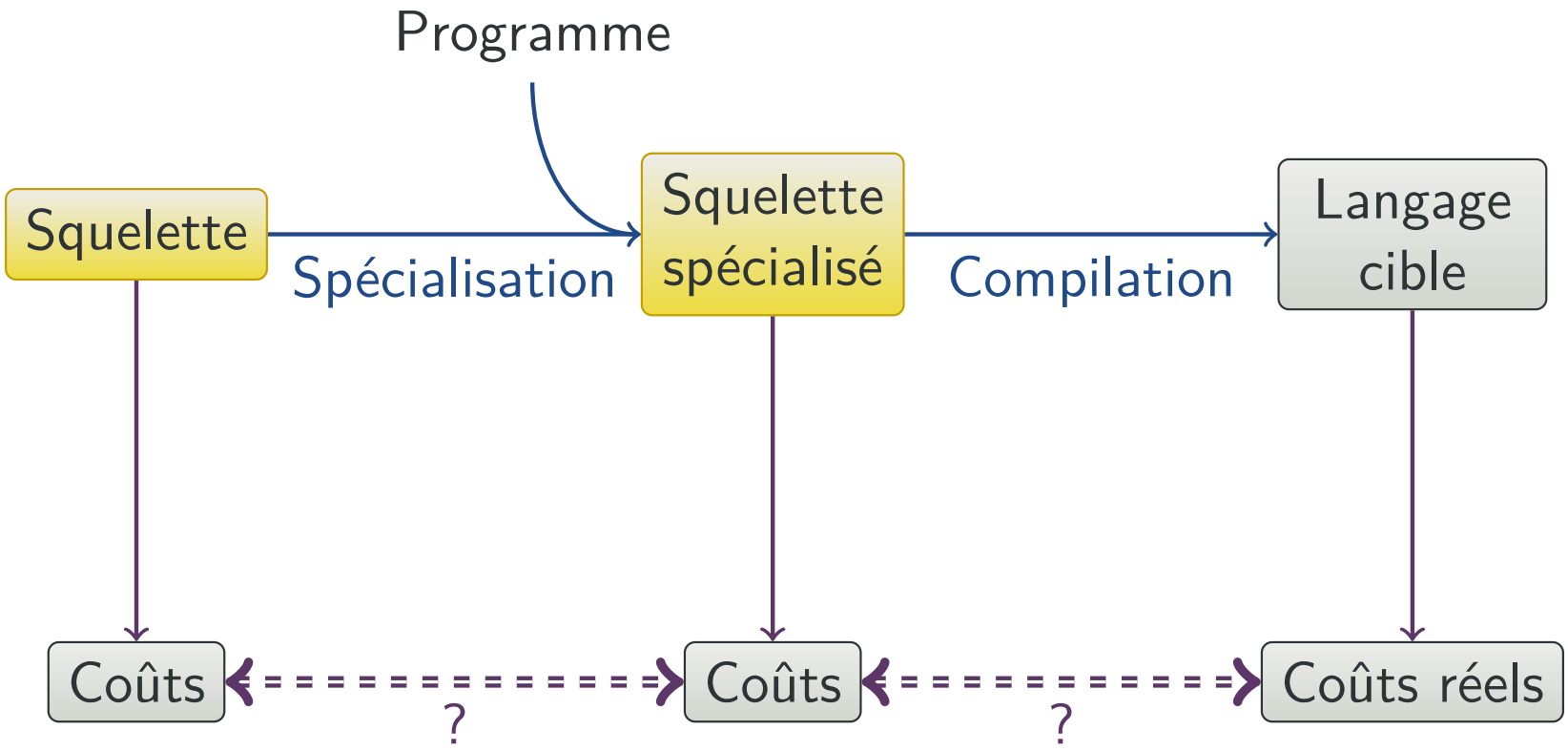
Squelettes : présent et futur



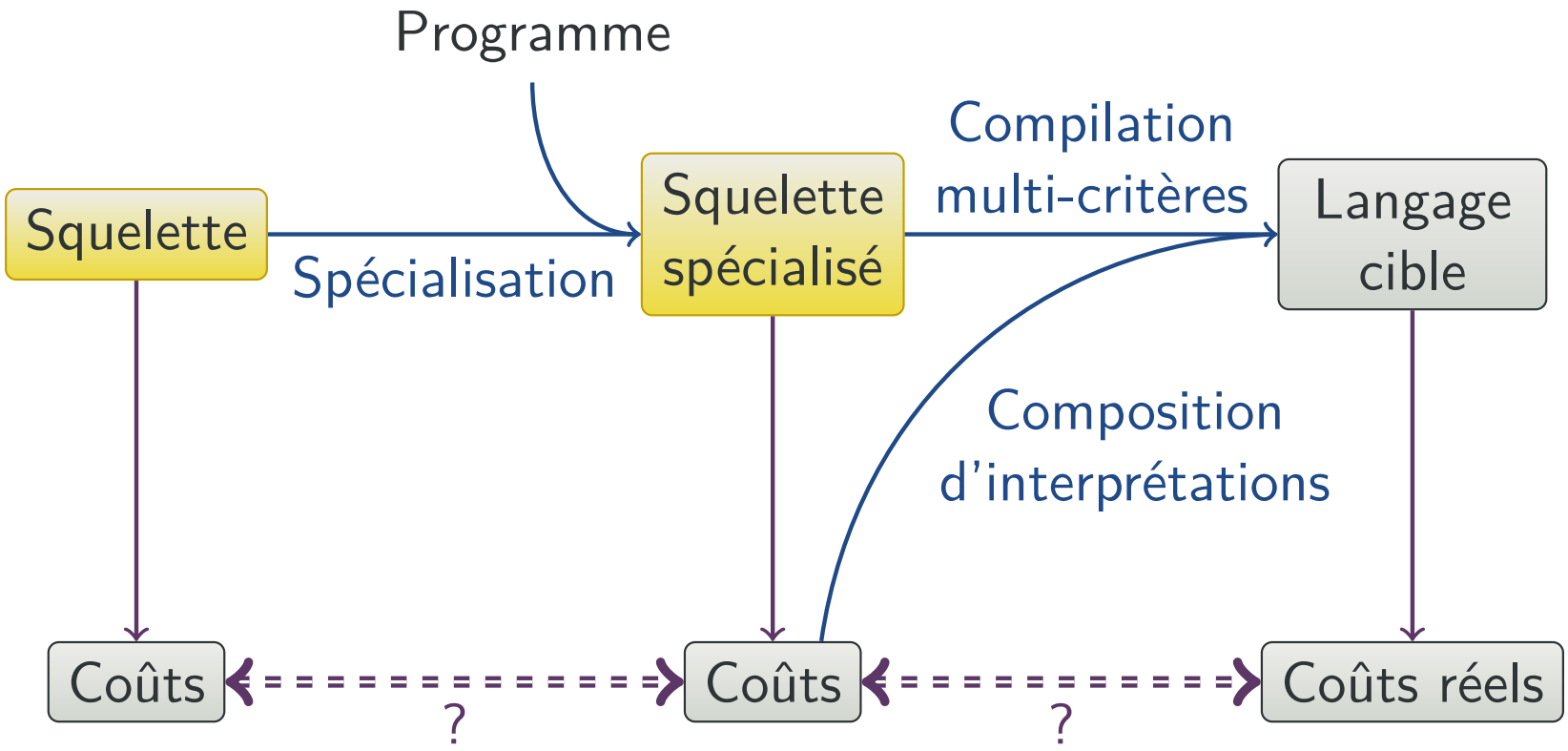
Compilation multi-critères avec les squelettes



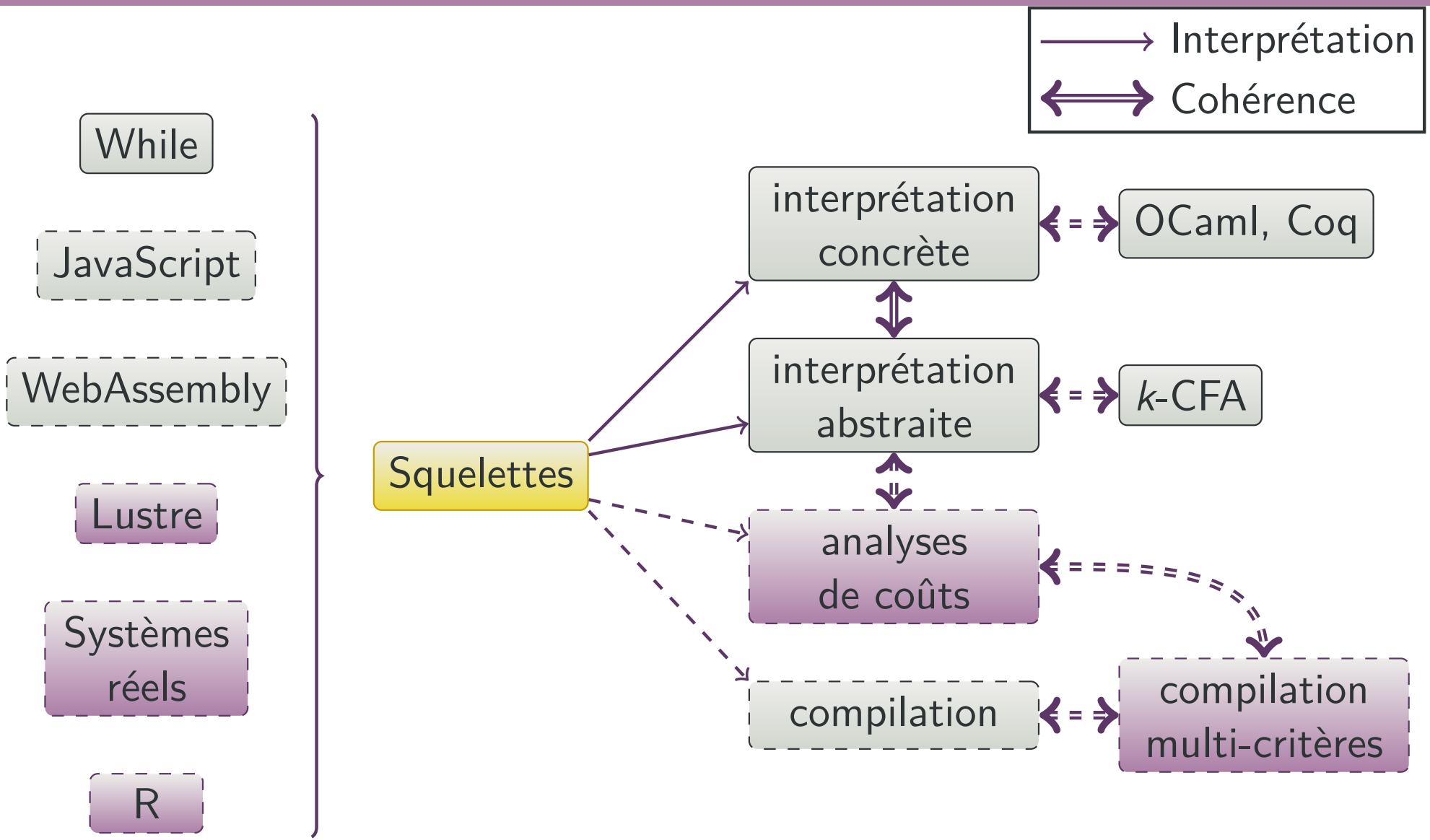
Compilation multi-critères avec les squelettes



Compilation multi-critères avec les squelettes



Conclusion


Interprétation
Cohérence
While
JavaScript
WebAssembly
Lustre
Systèmes réels
R
Squelettes
interprétation concrète
interprétation abstraite
analyses de coûts
compilation
OCaml, Coq
k-CFA
compilation multi-critères

1. Les langages populaires sont complexes
2. Formaliser ces langages en Coq
3. Squelettes : une syntaxe pour la sémantique
4. Projet de recherche

Bonus

1. Introduction à R,
2. R: Un langage de programmation paresseux,
3. CoqR,
4. Correspondance ligne à ligne,
5. Preuve dans JSCert,
6. Définition formelle des squelettes,
7. Interprétation concrète,
8. Interprétation abstraite,
9. Interprétation de coûts,
10. Cohérences d'interprétations,
11. Composer des interprétations,
12. Parallélisme et squelettes.

R: Un langage de vecteurs

```r
v <- c(10, 12, 14, 11, 13)
v[1]                                # Retourne 10
```

R: Un langage de vecteurs

```r
v <- c(10, 12, 14, 11, 13)
v[1]                          # Retourne 10
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(14, 13, 10)
```

R: Un langage de vecteurs

```r
v <- c(10, 12, 14, 11, 13)
v[1]                          # Retourne 10
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(14, 13, 10)
v[-2]                         # Retourne c(10, 14, 11, 13)
```

R: Un langage de vecteurs

```r
v <- c(10, 12, 14, 11, 13)
v[1]                          # Retourne 10
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(14, 13, 10)
v[-2]                         # Retourne c(10, 14, 11, 13)
v[-indices]                   # Retourne c(12, 11)
```

R: Un langage de vecteurs

```r
v <- c(10, 12, 14, 11, 13)
v[1]                          # Retourne 10
indices <- c(3, 5, 1)
v[indices]                    # Retourne c(14, 13, 10)
v[-2]                         # Retourne c(10, 14, 11, 13)
v[-indices]                   # Retourne c(12, 11)
v[c(FALSE, TRUE, FALSE)]      # Retourne c(12, 13)
```

R: Un langage de vecteurs

```r
v <- c(10, 12, 14, 11, 13)
v[1]                              # Retourne 10
indices <- c(3, 5, 1)
v[indices]                        # Retourne c(14, 13, 10)
v[-2]                             # Retourne c(10, 14, 11, 13)
v[-indices]                       # Retourne c(12, 11)
v[c(FALSE, TRUE, FALSE)]          # Retourne c(12, 13)
v[v > 11]                         # Retourne c(12, 14, 13)
```

R: Un langage de programmation paresseux

```
1  f <- function (x, y = x) {
2      x <- 1
3      y
4      x <- 2
5      y
6  }
7  f (3)
```

R: Un langage de programmation paresseux

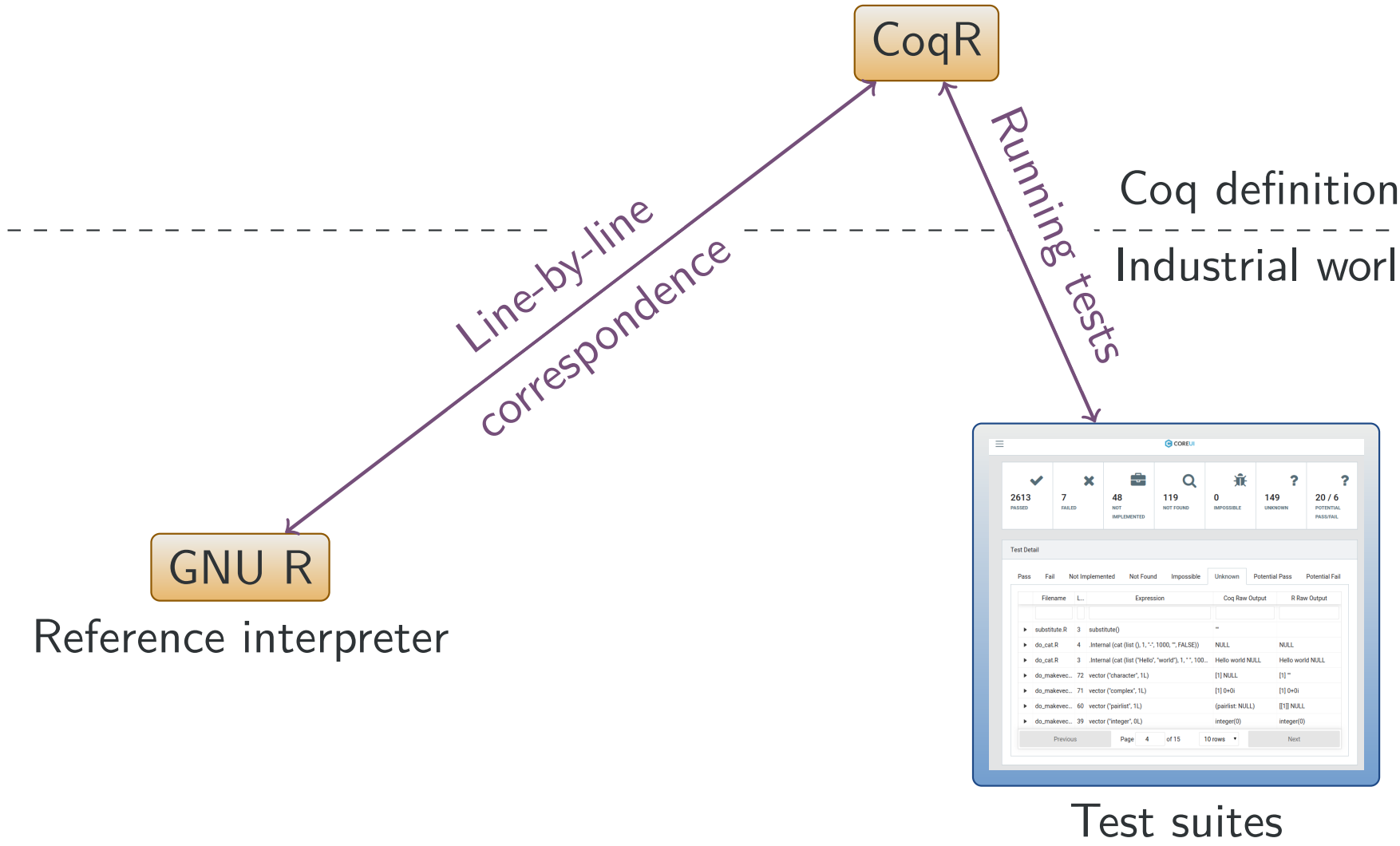
```r
f <- function (x, y = x) {
    x <- 1
    y
    x <- 2
    y
}
f (3)                                   # Retourne 1
```

R: Un langage de programmation paresseux

```r
f <- function (x, y = x) {
    x <- 1
    y
    x <- 2
    y
}
f (3)                                   # Retourne 1
```

```r
f <- function (x, y) if (x == 1) y
f (1, a <- 1)
a                                       # Retourne 1
f (0, b <- 1)
b                                       # Erreur
```

CoqR : faire confiance à R



https://github.com/Mbodin/CoqR

Correspondance ligne à ligne

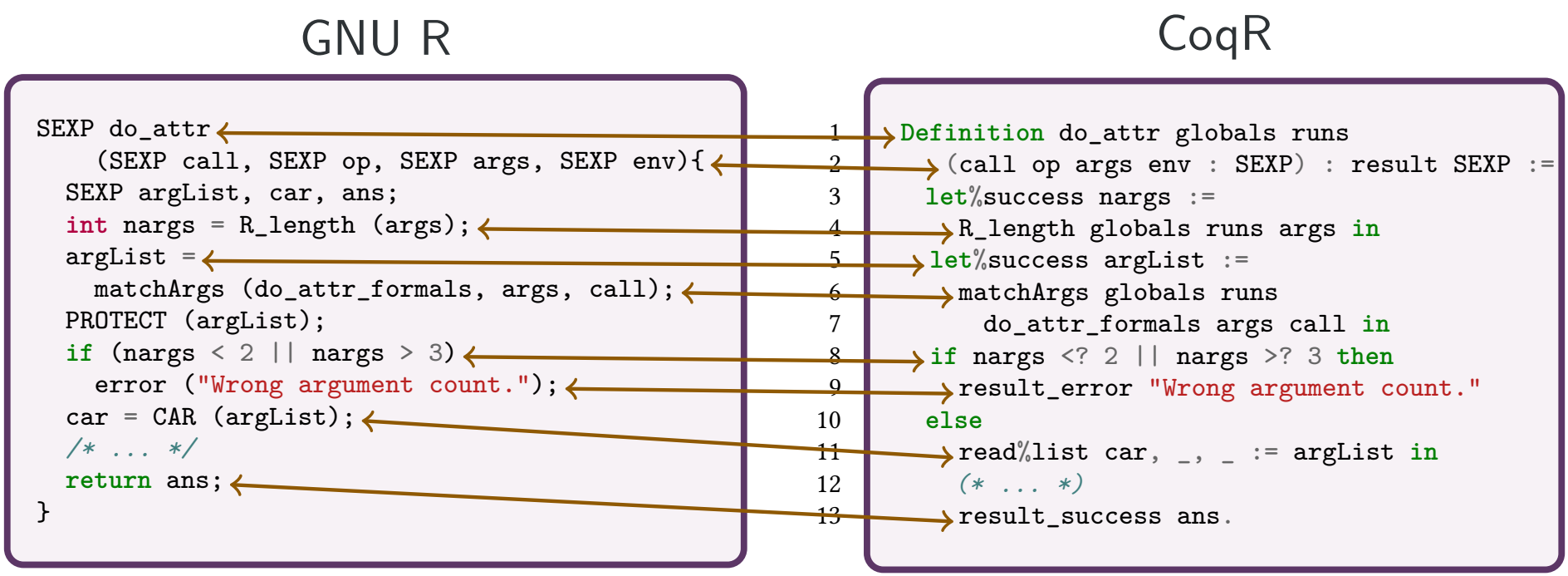
GNU R

```
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



Correspondance ligne à ligne

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.
```

ECMAScript

"s1 ; s2" is evaluated as follows.

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

JSCert

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

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

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

Preuve dans JSCert

```
Lemma one_plus_one_exec : forall S C,
  red_expr S C (expr_binary_op one binary_op_add one) (out_ter S (prim_number two)).
Proof.
  intros.
  eapply red_expr_binary_op.
   constructor.
   eapply red_spec_expr_get_value.
    eapply red_expr_literal. reflexivity.
   eapply red_spec_expr_get_value_1.
   eapply red_spec_ref_get_value_value.
  eapply red_expr_binary_op_1.
   eapply red_spec_expr_get_value.
    eapply red_expr_literal. reflexivity.
   eapply red_spec_expr_get_value_1.
   eapply red_spec_ref_get_value_value.
  eapply red_expr_binary_op_2.
  eapply red_expr_binary_op_add.
   eapply red_spec_convert_twice.
    eapply red_spec_to_primitive_pref_prim.
   eapply red_spec_convert_twice_1.
    eapply red_spec_to_primitive_pref_prim.
   eapply red_spec_convert_twice_2.
  eapply red_expr_binary_op_add_1_number.
   simpl. intros [A|A]; inversion A.
   eapply red_spec_convert_twice.
    eapply red_spec_to_number_prim. reflexivity.
   eapply red_spec_convert_twice_1.
    eapply red_spec_to_number_prim. reflexivity.
   eapply red_spec_convert_twice_2.
  eapply red_expr_puremath_op_1. reflexivity.
Qed.
```

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

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

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

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

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

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

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

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

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

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

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

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

Interprétation concrète

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

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

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

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

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

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

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

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

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

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

Interprétation abstraite

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

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

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

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

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

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

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

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

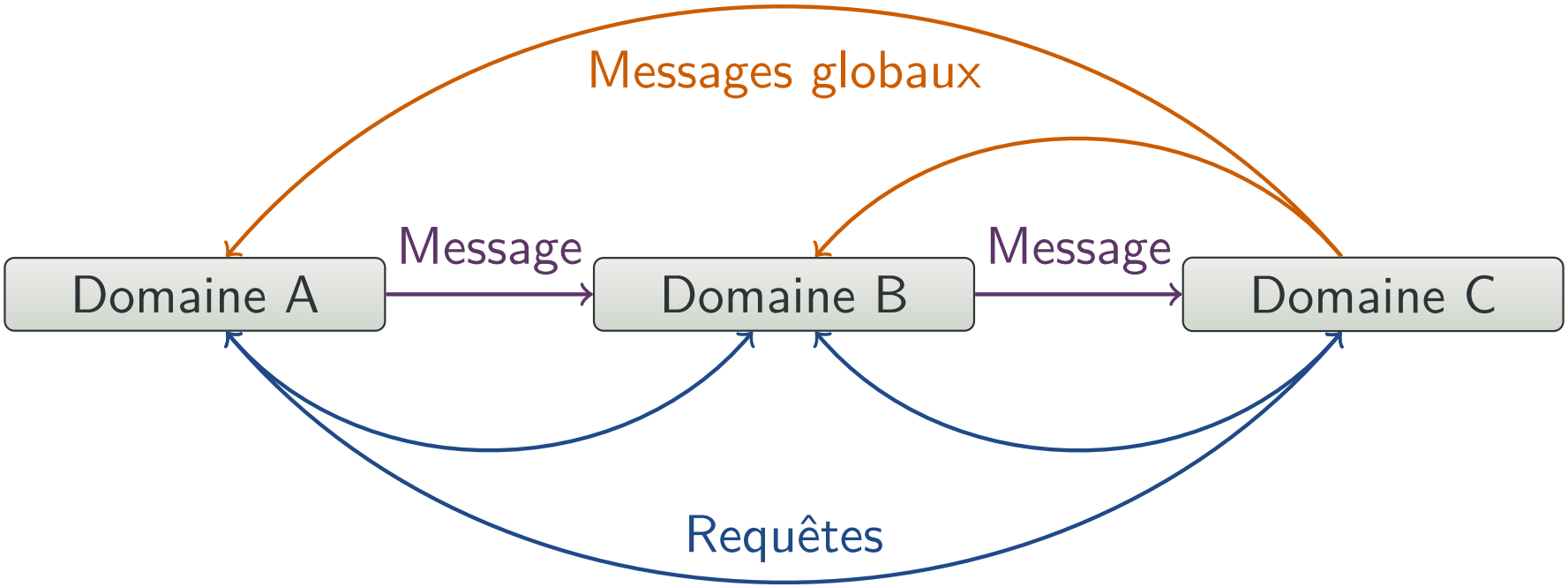
$$
\begin{aligned}
OKst\left((E,T),\left(E^\sharp,T^\sharp\right)\right) ::= \; & dom\,(E) = dom\left(E^\sharp\right) \\
& \wedge \forall x.\; E[x] \in \gamma\left(E^\sharp[x]\right) \\
& \wedge \forall (v_1,t,v_2) \in T, (v_1^\sharp,t,v_2^\sharp) \in T^\sharp. \\
& \quad v_1 \in \gamma\left(v_1^\sharp\right)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

Théorème

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

Composer des interprétations

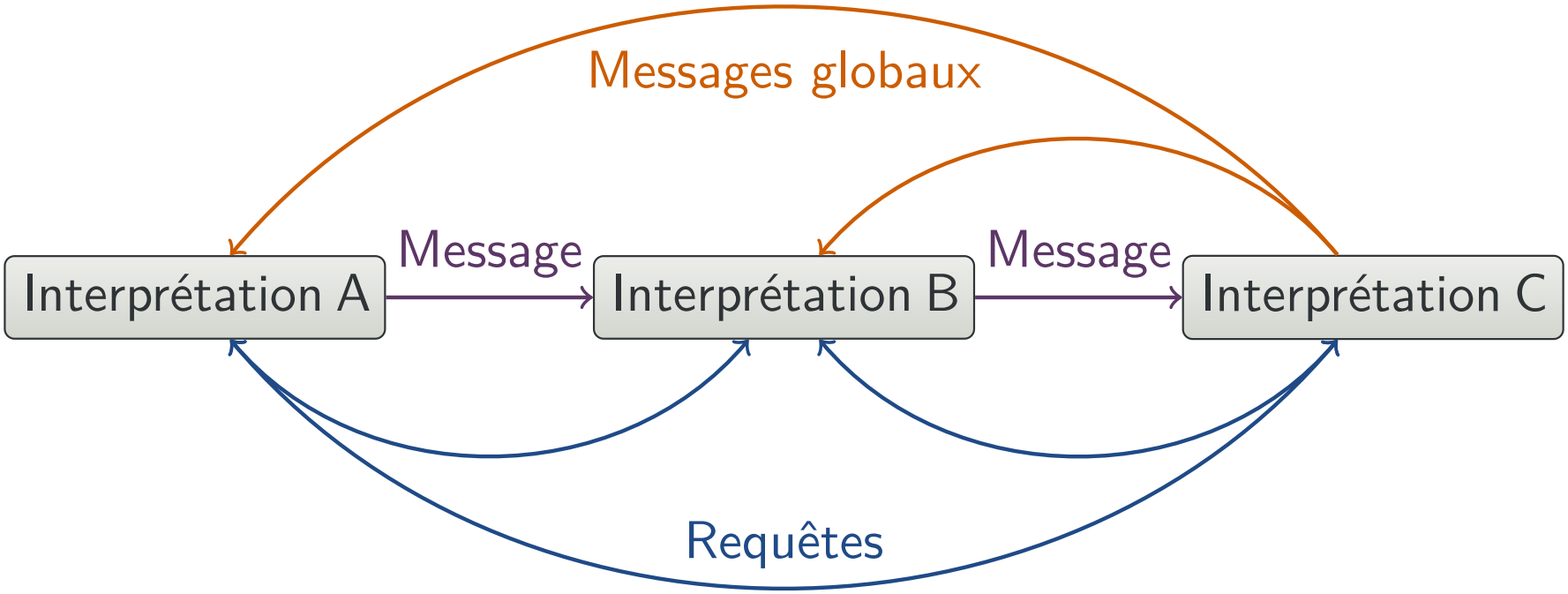
Idée de l'interprétation abstraite, qui fonctionne avec les squelettes.



Comment la généraliser à plusieurs interprétations ?

Composer des interprétations

Idée de l'interprétation abstraite, qui fonctionne avec les squelettes.



Comment la généraliser à plusieurs interprétations ?

$$[\sigma, (x = 1; a = y) \parallel (y = 1; b = x)] := \begin{bmatrix} let\ t_0 = \texttt{makeThread}\,(\sigma)\ in \\ let\ t_1 = \texttt{write}\,(t_0, \texttt{"x"}, 1)\ in \\ let\ t_2 = \texttt{write}\,(t_1, \texttt{"a"}, \texttt{"y"})\ in \\ let\ t'_0 = \texttt{makeThread}\,(\sigma)\ in \\ let\ t'_1 = \texttt{write}\,(t'_0, \texttt{"y"}, 1)\ in \\ let\ t'_2 = \texttt{write}\,(t'_1, \texttt{"b"}, \texttt{"x"})\ in \\ \texttt{parallel}\,(t_2, t'_2) \end{bmatrix}$$

1. Introduction à R,
2. R: Un langage de programmation paresseux,
3. CoqR,
4. Correspondance ligne à ligne,
5. Preuve dans JSCert,
6. Définition formelle des squelettes,
7. Interprétation concrète,
8. Interprétation abstraite,
9. Interprétation de coûts,
10. Cohérences d'interprétations,
11. Composer des interprétations,
12. Parallélisme et squelettes.

1. Les langages populaires sont complexes
2. Formaliser ces langages en Coq
3. Squelettes : une syntaxe pour la sémantique
4. Projet de recherche