Formalisation de langages de programmation en Coq

Martin Bodin

29 juin

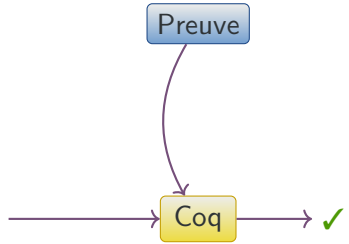
martin.bodin@inria.fr

L'assistant de preuve Coq pour prouver des théorèmes

L'assistant de preuve Coq pour prouver des propriétés d'une fonction

$f : \mathbb{R} \to \mathbb{R}$

$\forall x, f(x) > 0$

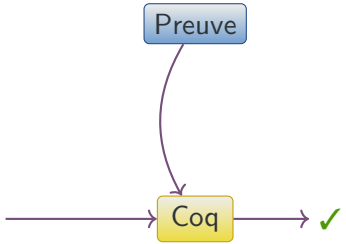


L'assistant de preuve Coq pour prouver des propriétés d'un analyseur

$$f : \texttt{code} \to \{\checkmark, \times\}$$

$$P \subseteq \texttt{code}$$

$$\forall c, f(c) = \checkmark \implies P(c)$$



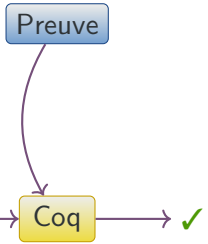
L'assistant de preuve Coq pour prouver des propriétés d'un analyseur

$$f : \texttt{code} \to \{\checkmark, \times\}$$

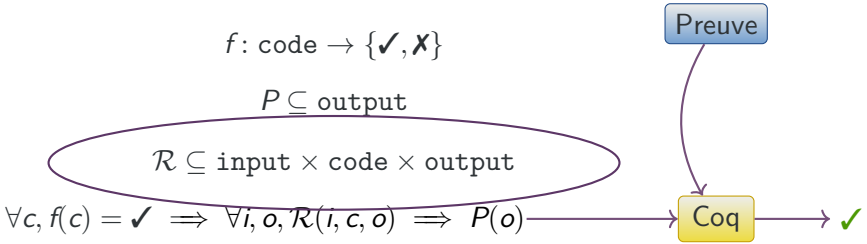
$$P \subseteq \texttt{output}$$

$$\mathcal{R} \subseteq \texttt{input} \times \texttt{code} \times \texttt{output}$$

$$\forall c, f(c) = \checkmark \implies \forall i, o, \mathcal{R}(i, c, o) \implies P(o)$$



L'assistant de preuve Coq pour prouver des propriétés d'un analyseur



R est parfois surprenant

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

R est parfois surprenant

```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 est parfois surprenant

```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 est parfois surprenant

```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 est parfois surprenant

```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 est parfois surprenant

```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 est parfois surprenant

```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 est parfois surprenant

```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 est parfois surprenant

```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.

https://gitlab.inria.fr/mbodin1/math-coq-tests/-/raw/master/MathC2plus/coq/puissances.v