Théorème de Rice

Motivation → Théorème très général démontrant que une grande partie des théorèmes intéressants sont indécidables.

Définition : Soit A un algorithme (string → bool).
On appelle langage de A et
on note L(A) l'ensemble $L(A) = \{s \in string \mid A \text{ s renvoie true}\}$.

Exemple : let A (n : string) : bool = est_premier n
$L(A) = \{"2", "3", "5", \dots\}$.

Remarque : $A = string \to bool$

Définition : Soit P une propriété sur les algorithmes.
On dit que P est sémantique si :
$\forall A, B \in \mathcal{A}, L(A) = L(B) \Rightarrow (P(A) \Leftrightarrow P(B))$.

Exemples :
- P(A) : "Le langage de A contient le mot vide" est une propriété sémantique.

- $Q(A)$: "Le programme OCaml A contient au plus 10 lignes" n'est pas sémantique.
- $R(A)$: "$\emptyset \subseteq \mathcal{L}(A)$" est sémantique.

Définition : Soit P une propriété sur les algorithmes.
On dit que P est non triviale si
$\exists A, B \in \mathcal{A}$, $P(A) \wedge \neg P(B)$.

Théorème de Rice : Soit P une propriété sémantique non triviale sur les algorithmes.
Alors, le problème de savoir si A vérifie P pour un algorithme A d'entrée est indécidable.

Exemples : . P est non trivial et est sémantique donc $A \mapsto$ A vérifie P ? est indécidable.

. Q est non trivial mais n'est pas sémantique donc le théorème de Rice n'est pas applicable : ici $A \mapsto$ A vérifie Q ? est décidable.

. R est sémantique mais triviale : $A \to$ A vérifie R ? est décidable (fun _ -> true).

Donc : Il est important de bien vérifier que P est non triviale et sémantique.

Preuve : Soit P une propriété sémantique non triviale sur les algorithmes. On pose l'algorithme A défini par

let a s = while true do () done

et sA son code. Sans perte de généralité, on suppose $P(A)$ vraie donc il existe B telle que $\neg P(B)$ par non trivialité de P.

On souhaite faire une réduction depuis l'arrêt.

Par l'absurde : si $f : A \longrightarrow B$ est

$C \longmapsto C$ vérifie P !

décidable par un algorithme de code f : string → bool.

On pose :

let halts (s: string) (e : string) : bool =

f "let _ = eval {s} {e} in {sB}" <> fA

(Les "{x}" n'existent pas en OCaml mais c'est plus simple à écrire).

- Si eval s e termine, alors le langage de l'algorithme B' dont le code est "let _ = eval {s} {e} in {sA}" est le même que le langage $L(B)$. De plus, P est sémantique donc $P(B') \iff P(B)$, d'où halts renvoie bien true.

- Sinon, si eval se ne termine pas, alors $L(B') = L(A)$ donc $P(A) \Leftrightarrow P(B'')$: halts renvoie alors false.

Le fonction halts décide donc bien le problème de l'arrêt : absurde.