

Td n° 7

Décidabilité, Calculabilité

Hugo SALOU MPI*

Dernière mise à jour le 5 décembre 2022

1 Quelques problèmes décidables

1. Soit $f : \mathbb{R} \rightarrow \mathbb{R}$.
 - Si f admet un zéro, on pose $\mathcal{M} = \text{fun } s \rightarrow \text{true}$.
 - Si f n'admet pas un zéro, on pose $\mathcal{M} = \text{fun } s \rightarrow \text{false}$.Alors, \mathcal{M} décide ZERO_f .
2. Soit \mathcal{M} une machine, et soit $w \in \Sigma^*$.
 - Si \mathcal{M} se termine sur l'entrée w , alors on pose $\mathcal{M}' = \text{fun } s \rightarrow \text{true}$.
 - Si \mathcal{M} ne se termine pas sur l'entrée w , alors on pose $\mathcal{M}' = \text{fun } s \rightarrow \text{false}$.Alors, \mathcal{M}' décide $\text{ARRÊT}_{\mathcal{M},w}$.

3. Le problème est trivialement vrai. En effet, soit $M \in \mathcal{O}$, de la forme

```
1 let m (s: string): string =
2   (code)
```

On crée la machine \mathcal{N} ci-dessous.

```
1 let n (s: string): string =
2   if true then
3     (code)
4   else
5     (code)
```

On a $m \neq n$, mais $\mathcal{L}(m) = \mathcal{L}(n)$, donc le problème est vrai sur toute entrée et la fonction $\text{fun } s \rightarrow \text{true}$ répond au problème.

2 Sérialisation de types énumérés

1. Pour sérialiser une liste, on utilise la fonction `serialize_couple` définie dans le cours.

```
1 let rec serialize_liste (t: 'a list) (serialize: 'a ->
2   string): string =
3   match s with
4   | [] -> ""
5   | x :: q -> serialize_couple (serialize x) (
6     serialize_liste t serialize)
```

CODE 1 – Sérialisation de listes

2. Soit $(\varphi_{i,j})_{\substack{i \in [1,m] \\ j \in [1,n_i]}}$ sérialisant le type $t_{i,j}$. Soit

$$\varphi_i : (x_1, \dots, x_{n_i}) \mapsto "(^{\wedge} \varphi_{i,1}(x_1) ^{\wedge}), (^{\wedge} \dots ^{\wedge} \varphi_{i,n_i}(x_{n_i}) ^{\wedge})"$$

```
1 let rec serialize_enum (e : enumeration) : string =
2   let rec aux (i: int) (j: int)
3   match e with
4   | C1(e1,1, ..., e1,n1) ->
5     serialize_couple (1, φ1(e1, ..., e1,n1))
6   | ...
7   | Cm(em,1, ..., em,nm) ->
8     serialize_couple (m, φm(e1, ..., enm))
```

CODE 2 – Sérialisation de types énumérés

3 Stabilité de la classe des langages décidables

1. Soit L un langage fini. On pose donc $L = \{w_1, \dots, w_n\}$. On code donc la fonction ci-dessous.

```
1 let decideL (w: string): bool =
2   match w with
3   | w1 -> true
4   | ...
5   | wn -> true
6   | _ -> false
```

CODE 3 – Fonction décidant d'un langage fini

Autre preuve : tout langage fini est régulier, et tout langage régulier est reconnaissable, et est donc décidable.

- Soient L_1 et L_2 deux langages décidables. Soit decide_{L_1} et decide_{L_2} décidant respectivement L_1 et L_2 . On code la fonction $\text{decide}_{L_1 \cdot L_2}$:

```

1 let decideL1·L2 (w: string) : bool =
2   let m = String.length w in
3   let rep = ref false in
4   for i = 0 to n - 1 do
5     if decideL1 w[[0,i]] && decideL2 w[[i+1,n-1]] then
6       rep := true
7   done;
8   !rep

```

CODE 4 – Fonction décidant d’une concaténation de langages décidables

-
-
-
- Tout langage singleton est décidable. Soit $L_{\text{ARRÊT}}$ l’ensemble

$$L_{\text{ARRÊT}} = \{\text{serialise_couple } M w \mid M \text{ s’arrête sur } w\}.$$

Le langage $L_{\text{ARRÊT}}$ est indécidable. Mais, $L_{\text{ARRÊT}} = \bigcup_{x \in L_{\text{ARRÊT}}} \{x\}$, est une union dénombrable.

4 Non monotonie du caractère décidable des langages

- Montrer qu’il existe trois langages A , B et C tels que $A \subseteq B \subseteq C$, que A soit décidable, B indécidable et C décidable.
- Montrer qu’il existe A , B et C trois langages tels que $A \subseteq B \subseteq C$, que A soit indécidable, B décidable, et C indécidable.

- On pose $A = \emptyset$, et $C = \Sigma^*$, et $B = \{(M, w) \mid M \text{ s’arrête sur } w\}$.
- Soit $w \in \Sigma^*$. On pose

$$L_0 = \{\text{serialise_couple}(w, M) \mid M \in L_{\text{ARRÊTUNIV}}^1\}.$$

Montrons que le langage L_0 est indécidable. Soit $M \in \mathcal{O}$ une entrée du problème ARRÊTUNIV .

$$\begin{aligned} \text{serialise_couple}(w, M) \in L_0 &\iff M \text{ s’arrête sur toutes ses entrées} \\ &\iff M \in \text{ARRÊTUNIV}^+. \end{aligned}$$

En effet, la fonction f de cette réduction est définie comme ci-dessous.

```

1 let f (M : string) : string =
2   serialise_couple w M

```

Par réduction de ARRÊTUNIV à APPARTIENT_{L_0} , le problème APPARTIENT_{L_0} est indécidable. On pose ensuite

$$L_1 = \{\text{serialise_couple}(M, w) \mid w \in \Sigma^*, M \in L_{\text{ARRÊTUNIV}}\}.$$

Soit $M \in \mathcal{O}$ une entrée du problème ARRÊTUNIV . La fonction g de cette réduction est définie comme ci-dessous.

```

1 let g (M: string) : string =
2   serialise_couple M "a"

```

$$\begin{aligned} g(M) \in L_1 &\iff \begin{cases} \text{"a"} \in \Sigma^* \\ M \in \text{ARRÊTUNIV}^+ \end{cases} \\ &\iff M \in \text{ARRÊTUNIV}^+ \end{aligned}$$

On pose $w = \text{"fun } s \rightarrow \text{true."}$ On a $L_0 \subseteq L \subseteq L_2$ où

$$L = \{\text{serialise_couple}(\text{"fun } s \rightarrow \text{true"}, w) \mid w \in \Sigma^*\}.$$

Le langage L est décidable. En effet, le code ci-dessous décide du problème APPARTIENT_L .

```

1 let decideL (w: string) : bool =
2   let m = String.length w in
3   w.[n - 1] = ' ' && w[0,15] = "(fun s → true)"

```

5 Réduction

1. Soit m, w les entrées du problème de l'ARRÊT. On fabrique la sérialisation de machine

“fun s → execute {{m}} {{w}}.”

Notons $M'_{m,w}$ cette machine. On a

$$\begin{aligned}
M'_{m,w} \in \text{ARRÊTUNIV}^+ &\iff \forall x \in \Sigma^*, \text{ execute } m \text{ } w \text{ se termine} \\
&\iff \text{ execute } m \text{ } w \text{ se termine} \\
&\iff (m, w) \in \text{ARRÊT}^+
\end{aligned}$$

Ainsi, on crée la réduction.

```

1 let reduction (s: string): string =
2   let (m, w) = deserialise_couple s in
3   "fun s → execute " ^ m ^ " " ^ w

```

CODE 5 – Réduction de ARRÊTUNIV au problème de l'ARRÊT

Ainsi, ARRÊTUNIV est indécidable par réduction.

2. Réduisons le problème ARRÊT à ARRÊTSIMULT. Soit m, w les données du problème de l'ARRÊT. Posons les machines $M_{m,w} = \text{“fun } x \rightarrow \text{execute } \{\{m\}\} \{\{w\}\} \text{”}$, et $N : \text{“fun } x \rightarrow 3 \text{”}$. On a

$$\begin{aligned}
&\text{serialise_couple}(M_{m,w}, N) \in \text{ARRÊT}^+ \\
&\iff (\forall x \in \Sigma^*, \text{ execute } M_{m,w} \text{ } x \text{ termine} \iff \text{ execute } N \text{ } x \text{ termine}) \\
&\iff (\text{execute } \{\{m\}\} \{\{w\}\} \text{ termine} \iff V) \\
&\iff \text{execute } \{\{m\}\} \{\{w\}\} \text{ termine} \\
&\iff (m, w) \in \text{ARRÊT}^+.
\end{aligned}$$

Autre possibilité : réduisons ARRÊTUNIV à ARRÊTSIMULT. On pose $N : \text{“fun } x \rightarrow 2 \text{”}$, et on a

$$\begin{aligned}
&\text{serialise_couple}(M, N) \in \text{ARRÊTSIMULT}^+ \\
&\iff (\forall x \in \Sigma^*, \text{ execute } M \text{ } x \text{ termine} \iff \text{ execute } N \text{ } x \text{ termine}) \\
&\iff \forall x \in \Sigma^*, \text{ execute } M \text{ } x \text{ termine} \\
&\iff M \in \text{ARRÊTUNIV}^+.
\end{aligned}$$

3. Réduisons ARRÊT à RÉGULIER. Soit (m, w) une entrée du problème de l'ARRÊT.

```

1 let reconnait_an_bn (w: string): bool =
2   let n = String.length w in
3   if n mod 2 = 1 then false
4   else begin
5     let ok = ref true in
6     for i = 1 to n / 2 do
7       if w.[i] = 'a' then ok := false
8     done;
9     for i = (n / 2) + 1 to n do
10      if w.[i] = 'b' then ok := false
11    done;
12    !ok
13  end

```

CODE 6 – Machine reconnaissant le langage $\{a^n \cdot b^n \mid n \in \mathbb{N}\}$

On considère la fonction de réduction ci-dessous.

```

1 let f (e: string): string =
2   let (m, w) = deserialise_couple e in
3   "fun s → execute {{m}} {{w}}; reconnait_an_bn {{w}}"

```

En notant $(m, w) = \text{deserialise_couple}(e)$,

$$(f e) \in \text{RÉGULIER}^+ \iff \text{“fun } s \rightarrow \text{execute}\{\{m\}\}\{\{w\}\}\text{”}.$$

Or,

— si $w \xrightarrow{m} \circ$, alors $\forall s \in \Sigma^*$, $s \xrightarrow{m} \circ$, donc $\mathcal{L}(m) = \emptyset$.

— si $w \xrightarrow{m} w'$, avec $w' \in \Sigma^*$, alors

— si $s \xrightarrow{m} \text{true} \iff s \in \{a^n \cdot b^n \mid n \in \mathbb{N}\}$,

— si $s \xrightarrow{m} \text{false} \iff s \notin \{a^n \cdot b^n \mid n \in \mathbb{N}\}$,

Et donc, $\mathcal{L}(m) = \{a^n \cdot b^n \mid n \in \mathbb{N}\}$.

On a donc

$$\begin{aligned} (f e) \in \text{RÉGULIER}^+ &\iff w \xrightarrow{m} \circ \\ &\iff (m, w) \in \text{ARRÊT}^- \\ &\iff (m, w) \in \text{CoARRÊT}^+ \end{aligned}$$

où le problème CoARRÊT est défini comme

$$\text{CoARRÊT} : \begin{cases} \text{Entrée} & : (m, w) \\ \text{Sortie} & : \text{A-t-on } w \xrightarrow{m} \circ \end{cases}$$

On a $\text{CoARRÊT}^- = \text{ARRÊT}^+$, et $\text{CoARRÊT}^+ = \text{ARRÊT}^-$. Le problème CoARRÊT est indécidable par stabilité des problèmes décidables par complémentaire. On conclut par réduction de CoARRÊT à RÉGULIER.

4. Réduisons ARRÊT à ARRÊT_w. Soit (w, M) une entrée du problème ARRÊT. Fabriquons la machine $M_w : \text{“fun } s \rightarrow \text{execute } M w \text{”}$. On a

$$\begin{aligned} M_w \in \text{ARRÊT}_w^+ &\iff (\forall x \in \Sigma^*, \text{execute } M w \text{ se termine}) \\ &\iff \text{execute } M w \text{ se termine} \\ &\iff (M, w) \in \text{ARRÊT}^+ \end{aligned}$$

Par réduction, le problème ARRÊT_w est indécidable.

5. Réduisons ARRÊTUNIV à ARRÊTEXISTE. Soit M une entrée du problème ARRÊTEXISTE.

6 Amélioration de code

1.

```
1 let mort (s: string): string =
2   let f (a: int) = a in
3   "sortie"
```

CODE 7 – Fonction morte

2. On réalise une réduction du problème ARRÊTUNIV au problème CODEMORT. Soit \mathcal{M} une entrée du problème ARRÊTUNIV. Fabriquons l'entrée M du problème CODEMORT, comme montré ci-dessous.

```
1 let M (s: string): string =
2   execute M s;
3   let f (a: string) = a in
4   f x
```

CODE 8 – Réduction de ARRÊTUNIV à CODEMORT

Ainsi, avec cette construction de la machine M , on définit u comme `execute M s`, on définit d comme `let f (a: string) = a in`, et on définit v comme `f x`. Alors,

$$\begin{aligned} M \in \text{CODEMORT}^+ &\iff \mathcal{L}(u \cdot d \cdot v) = \mathcal{L}(u \cdot v) \\ &\iff \forall s \in \Sigma^*, \text{ la ligne 4 est atteinte} \\ &\iff \forall s \in \Sigma^*, \text{execute } M s \text{ se termine} \\ &\iff \forall s \in \Sigma^*, M \text{ se termine sur l'entrée } s \\ &\iff \mathcal{M} \in \text{ARRÊTUNIV}^+. \end{aligned}$$

Or, comme ARRÊTUNIV est indécidable, CODEMORT aussi.

3.

```

1 let f (s: string): string =
2   let x = ref 3 in
3   let y = int_of_string s in
4   string_of_int (!x + y)
5
6 let f (s: string): string =
7   let y = int_of_string s in
8   string_of_int (3 + y)

```

CODE 9 – Variable constante

4. Soit M une entrée du problème CODEMORT. Fabriquons l'entrée (M', \mathcal{V}) du problème P de détection de variables constantes. Soit F l'ensemble des "fonctions locales" de M .² Au début de la machine M' , on définit, pour chaque fonction locale $f \in F$, la variable x_f comme `let x_f = ref 0 in`. On pose \mathcal{V} l'ensemble de ces nouvelles variables :

$$\mathcal{V} = \{x_f \mid f \in F\}.$$

Puis, on transforme chaque définition de fonction locale $f \in F$ en

```

1 let f ⟨arguments de f⟩ =
2   incr  $x_f$ ;
3   ⟨code original de f⟩

```

Ainsi,

$$\begin{aligned}
(M', \mathcal{V}) \in P^+ &\iff \exists x_f \in \mathcal{V}, \text{ la variable } x_f \text{ n'est pas modifiée} \\
&\iff \exists f \in F, \text{ la fonction } f \text{ n'est pas appelée} \\
&\iff \exists f \in F, f \text{ est une fonction morte} \\
&\iff M \in \text{CODEMORT}^+.
\end{aligned}$$

Or, comme le problème CODEMORT est indécidable, le problème de détection de variables constantes l'est aussi.

7 Théorème de Rice

8 Un changement de modèle de calcul

1. Pour toute machine M , on considère la *super-machine* \mathcal{M}_M suivante :

```

1 let arret ( $n_V$ : int) (x: string): bool =
2   let ( $M, w$ ) = deserialise_couple x in
3   decideARRÊTÉTAPES  $M$   $w$   $n_V$ 

```

CODE 10 – *Super-machine* résolvant le problème de l'ARRÊT sur une machine classique

où la fonction `decideARRÊTÉTAPES` décide du problème

$$\text{ARRÊTÉTAPES} : \begin{cases} \text{Entrée} & : M \in \mathcal{O}, w \in \Sigma^*, n \in \mathbb{N} \\ \text{Sortie} & : M \text{ se termine-t-elle sur } w \text{ en moins de } n \text{ étapes élémentaires?} \end{cases}$$

D'après le cours, ce problème est décidable. Ainsi, pour tout mot $w \in \Sigma^*$ et toute machine M ,

$$\begin{aligned}
w \xrightarrow{\mathcal{M}_M} V &\iff \exists n \in \mathbb{N}, \text{ arret } n (\text{serialise_couple } M w) = \text{true} \\
&\iff \exists n \in \mathbb{N}, M \text{ s'arrête en moins de } n \text{ étapes sur } w \\
&\iff M \text{ s'arrête sur l'entrée } w \\
&\iff (M, w) \in \text{ARRÊT}^+.
\end{aligned}$$

2. Par l'absurde, supposons le problème de l'arrêt pour les *super-machines* décidable. Soit `arret` une fonction décidant de ce problème. On considère la machine suivante.

2. S'il y a plusieurs fonctions ayant le même nom, on les numérote de telle sorte que leurs noms soient différents.

```
1 let paradoxe (n: int) (w: string): bool =
2   if arret n (serialise_couple w w) then
3     (while true do () done; true)
4   else false
```

CODE 11 – Programme paradoxe prouvant que le problème de l'arrêt des *super-machines* est indécidable

Soit S_{paradoxe} la sérialisation de la fonction paradoxe ci-dessous. Analysons l'exécution de $(\text{paradoxe } n \ S_{\text{paradoxe}})$. Soit $c = (\text{serialise_couple } S_{\text{paradoxe}} \ S_{\text{paradoxe}})$.