TD n° 7

# *Décidabilité, Calculabilité*

Hugo SALOU MPI*

Dernière mise à jour le 22 janvier 2023

# 1 Quelques problèmes décidables

1. Soit $f : \mathbb{R} \to \mathbb{R}$.
   — Si $f$ admet un zéro, on pose $\mathscr{M} =$ `fun s` $\to$ `true`.
   — Si $f$ n'admet pas un zéro, on pose $\mathscr{M} =$ `fun s` $\to$ `false`.

   Alors, $\mathscr{M}$ décide $\textsc{Zero}_f$.
2. Soit $\mathscr{M}$ une machine, et soit $w \in \Sigma^*$.
   — Si $\mathscr{M}$ se termine sur l'entrée $w$, alors on pose $\mathscr{M}' =$ `fun s` $\to$ `true`.
   — Si $\mathscr{M}$ ne se termine pas sur l'entrée $w$, alors on pose $\mathscr{M}' =$ `fun s` $\to$ `false`.

   Alors, $\mathscr{M}'$ décide $\textsc{Arrêt}_{\mathscr{M},w}$.
3. Le problème est trivialement vrai. En effet, soit $M \in \mathscr{O}$, de la forme

```
let m (s: string): string =
  ⟨code⟩
```

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

```
let n (s: string): string =
  if true then
    ⟨code⟩
  else
    ⟨code⟩
```

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

# 2 Sérialisation de types énumérés

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

```
let rec serialise_liste (t: 'a list) (serialise: 'a ->
    string): string =
  match s with
  | [] -> ""
  | x :: q -> serialise_couple (serialise x) (
      serialise_liste t serialise)
```

Code 1 – Sérialisation de listes

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

$$\varphi_i : (x_1, \dots, x_{n_i}) \longmapsto \text{“(”} \mathbin{\hat{}} \varphi_{i,1}(x_1) \mathbin{\hat{}} \text{“),(”} \mathbin{\hat{}} \dots \mathbin{\hat{}} \varphi_{i,n_i}(x_{n_i}) \mathbin{\hat{}} \text{“)”}.$$

```
let rec serialise_enum (e : enumeration) : string =
  let rec aux (i: int) (j: int)
  match e with
  | C_1(e_{1,1},...,e_{1,n_1}) ->
    serialise_couple (1, φ_1(e_1,...,e_{n_1}))
  | ...
  | C_m(e_{m,1},...,e_{m,n_m}) ->
    serialise_couple (m, φ_m(e_1,...,e_{n_m}))
```

Code 2 – Serialisation 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.

```
let decide_L (w: string): bool =
  match w with
  | w_1 -> true
  | ...
  | w_n -> true
  | _ -> 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.

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

```
let decide_{L1·L2} (w: string) : bool =
  let m = String.length w in
  let rep = ref false in
  for i = 0 to n - 1 do
    if decide_{L1} w|[[0,i]] && decide_{L2} w|[[i+1,n-1]] then
      rep := true
  done;
  !rep
```

Code 4 – Fonction décidant d'une concaténation de langages décidables

3.
4. Tout langage singleton est décidable. Soit $L_{\textsc{Arrêt}}$ l'ensemble

$$L_{\textsc{Arrêt}} = \{\texttt{serialise\_couple}\ M\ w \mid M \text{ s'arrête sur } w\}.$$

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

# 4 Non monotonie du caractère décidable des langages

1. *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.*
2. *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.*

1. On pose $A = \varnothing$, et $C = \Sigma^*$, et $B = \{(M, w) \mid M \text{ s'arrête sur } w\}$.
2. Soit $w \in \Sigma^*$. On pose

$$L_0 = \{\texttt{serialise\_couple}(w, M) \mid M \in L_{\textsc{ArrêtUniv}}{}^{1}\}.$$

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

$$\begin{aligned}\texttt{serialise\_couple}(w, M) \in L_0 &\iff M \text{ s'arrête sur toutes ses entrées}\\ &\iff M \in \textsc{ArrêtUniv}^+.\end{aligned}$$

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

```
let f (M : string) : string =
  serialise_couple w M
```

Par réduction de ArrêtUniv à $\textsc{Appartient}_{L_0}$, le problème $\textsc{Appartient}_{L_0}$ est indécidable. On pose ensuite

$$L_1 = \{\texttt{serialise\_couple}(M, w) \mid w \in \Sigma^*, M \in L_{\textsc{ArrêtUniv}}\}.$$

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

```
let g (M: string) : string =
  serialise_couple M "a"
```

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

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

$$L = \{\texttt{serialise\_couple}(\text{“}\texttt{fun}\ s \to \texttt{true}\text{”}, w) \mid w \in \Sigma^*\}.$$

Le langage $L$ est décidable. En effet, le code ci-dessous décide du problème $\textsc{Appartient}_L$.

```
let decide_L (w: string) : bool =
  let m = String.length w in
  w.[n - 1] = ')' && w|⟦0,15⟧ = "(fun s -> true)("
```

## 5 Réduction

1. Soit $\mathtt{m}, w$ les entrées du problème de l'Arrêt. On fabrique la sérialisation de machine

$$\text{“}\mathtt{fun\ s} \to \mathtt{execute}\ \{\{\mathtt{m}\}\}\ \{\{w\}\}.\text{”}$$

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

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

   Ainsi, on crée la réduction.

```
let reduction (s: string): string =
  let (m, w) = deserialise_couple s in
  "fun x -> 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 $\mathtt{m}, w$ les données du problème de l'Arrêt. Posons les machines $M_{\mathtt{m},w} = $ “$\mathtt{fun}\ x \to \mathtt{execute}\ \{\{\mathtt{m}\}\}\ \{\{w\}\}$”, et $N$ : “$\mathtt{fun}\ x\ \to 3.$” On a

$$\begin{aligned} &\mathtt{serialise\_couple}(M_{\mathtt{m},w}, N) \in \text{Arrêt}^+ \\ \iff &(\forall x \in \Sigma^*, \quad \mathtt{execute}\ M_{\mathtt{m},w}\ x \text{ termine} \iff \mathtt{execute}\ N\ x \text{ termine}) \\ \iff &(\mathtt{execute}\ \{\{\mathtt{m}\}\}\ \{\{w\}\} \text{ termine} \iff \boldsymbol{V}) \\ \iff &\mathtt{execute}\ \{\{\mathtt{m}\}\}\ \{\{w\}\} \text{ termine} \\ \iff &(\mathtt{m}, w) \in \text{Arrêt}^+. \end{aligned}$$

   Autre possibilité : réduisons ArrêtUniv à ArrêtSimult. On pose $N$ : “$\mathtt{fun}\ x \to 2,$” et on a

$$\begin{aligned} &\mathtt{serialise\_couple}(M, N) \in \text{ArrêtSimult}^+ \\ \iff &(\forall x \in \Sigma^*, \quad \mathtt{execute}\ M\ x \text{ termine} \iff \mathtt{execute}\ N\ x \text{ termine}) \\ \iff &\forall x \in \Sigma^*,\ \mathtt{execute}\ M\ x \text{ termine} \\ \iff &M \in \text{ArrêtUniv}^+. \end{aligned}$$

3. Réduisons Arrêt à Régulier. Soit $(\mathtt{m}, w)$ une entrée du problème de l'Arrêt.

```
let reconnait_an_bn (w: string): bool =
  let n = String.length w in
  if n mod 2 = 1 then false
  else begin
      let ok = ref true in
      for i = 1 to n / 2 do
        if w.[i] = 'a' then ok := false
      done;
      for i = (n / 2) + 1 to n do
        if w.[i] = 'b' then ok := false
      done;
      !ok
    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.

```
let f (e: string): string =
  let (m, w) = deserialise_couple e in
  "fun s -> execute {{m}} {{w}}; reconnait_an_bn {{w}}"
```

En notant $(\mathtt{m}, \mathtt{w}) = \mathtt{deserialise\_couple(e)}$,

$$(f\ \mathtt{e}) \in \textsc{Régulier}^+ \iff \text{“}\mathtt{fun}\ s \to \mathtt{execute}\{\{\mathtt{m}\}\}\{\{\mathtt{w}\}\}\text{”}.$$

Or,

— si $w \underset{\mathtt{m}}{\longrightarrow} \circlearrowleft$, alors $\forall s \in \Sigma^*$, $s \underset{\mathtt{m}}{\longrightarrow} \circlearrowleft$, donc $\mathscr{L}(\mathtt{m}) = \varnothing$.

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

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

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

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

On a donc

$$\begin{aligned}(f\ \mathtt{e}) \in \textsc{Régulier}^+ &\iff w \underset{\mathtt{m}}{\longrightarrow} \circlearrowleft \\ &\iff (\mathtt{m}, w) \in \textsc{Arrêt}^- \\ &\iff (\mathtt{m}, w) \in \textsc{CoArrêt}^+\end{aligned}$$

où le problème $\textsc{CoArrêt}$ est défini comme

$$\textsc{CoArrêt} : \begin{cases}\textbf{Entrée} & : (\mathtt{m}, w) \\ \textbf{Sortie} & : \text{A-t-on } w \underset{\mathtt{m}}{\longrightarrow} \circlearrowleft\end{cases}.$$

On a $\textsc{CoArrêt}^- = \textsc{Arrêt}^+$, et $\textsc{CoArrêt}^+ = \textsc{Arrêt}^-$. Le problème $\textsc{CoArrêt}$ est indécidable par stabilité des problèmes décidables par complémentaire. On conclut par réduction de $\textsc{CoArrêt}$ à $\textsc{Régulier}$.

4. Réduisons $\textsc{Arrêt}$ à $\textsc{Arrêt}_w$. Soit $(w, M)$ une entrée du problème $\textsc{Arrêt}$. Fabriquons la machine $M_w$ : “$\mathtt{fun}\ s \to \mathtt{execute}\ M\ w$.” On a

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

Par réduction, le problème $\textsc{Arrêt}_w$ est indécidable.

5. Réduisons $\textsc{ArrêtUniv}$ à $\textsc{ArrêtExiste}$. Soit $M$ une entrée du problème $\textsc{ArrêtExiste}$.

# 6 Amélioration de code

1.

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

Code 7 – Fonction morte

2. On réalise une réduction du problème $\textsc{ArrêtUniv}$ au problème $\textsc{CodeMort}$. Soit $\mathscr{M}$ une entrée du problème $\textsc{ArrêtUniv}$. Fabriquons l'entrée $M$ du problème $\textsc{CodeMort}$, comme montré ci-dessous.

```
let M (s: string): string =
  execute ℳ s;
  let f (a: string) = a in
  f x
```

Code 8 – Réduction de $\textsc{ArrêtUniv}$ à $\textsc{CodeMort}$

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

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

Or, comme ArrêtUniv est indécidable, CodeMort aussi.

3.

```
let f (s: string): string =
  let x = ref 3 in
  let y = int_of_string s in
  string_of_int (!x + y)

let f (s: string): string =
  let y = int_of_string s in
  string_of_int (3 + y)
```

Code 9 – Variable constante

4. Soit $M$ une entrée du problème CodeMort. Fabriquons l'entrée $(M', \mathscr{V})$ 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 $\mathtt{f} \in F$, la variable $\mathtt{x_f}$ comme `let` $\mathtt{x_f}$ `= ref 0 in`. On pose $\mathscr{V}$ l'ensemble de ces nouvelles variables :

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

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

```
let f ⟨arguments de f⟩ =
  incr x_f;
  ⟨code original de f⟩
```

Ainsi,

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

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* $\mathscr{M}_M$ suivante :

```
let arret (n_V: int) (x: string): bool =
  let (M,w) = deserialise_couple x in
  decide_ArrêtÉtapes M w n_V
```

Code 10 – *Super-machine* résolvant le problème de l'Arrêt sur une machine classique

où la fonction $\mathtt{decide}_{\textsc{ArrêtÉtapes}}$ décide du problème

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

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

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

$$\begin{aligned} w \xrightarrow[\mathscr{M}_M]{} \boldsymbol{V} &\iff \exists n \in \mathbb{N},\ \texttt{arret}\ n\ (\texttt{serialise\_couple}\ M\ w) = \texttt{true} \\ &\iff \exists n \in \mathbb{N},\ M \text{ s'arrête en moins de } n \text{ étapes sur } w \\ &\iff M \text{ s'arrête sur l'entrée } w \\ &\iff (M, w) \in \textsc{Arrêt}^+. \end{aligned}$$

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.

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

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

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