Chapitre 10 : Décidabilité

February 7, 2025

Au programme : concepts à comprendre, démonstration à connaître !

1 Problème de décision et décidabilité

Définition 10.1 - *problème de décision*

Un *problème de décision* est un problème dont la réponse attendue est binaire : Vrai ou Faux. Plus précisément, un problème $\mathcal{P}$ est la donnée de :

- I un ensemble d'instances
- S un ensemble de solutions : l'union des solutions pour chaque instance.
- $f : I \to S$ ou pour i une instance, $f(i)$ est la réponse attendue pour l'instance i.

Pour un problème de décision, $S = \{\text{Vrai}, \text{Faux}\}$. On appelle la fonction f *fonction de prédicat* du problème $\mathcal{P}$ de décision.

$\mathcal{P}$ peut aussi être défini à l'aide d'un sous-ensemble P de $I \times S$ tel que :

$$(i, s) \in P \Leftrightarrow s \text{ solution de } \mathcal{P} \text{ pour l'instance } i$$

Pour $\mathcal{P}$ un problème de décision, défini par $f : I \to \{\text{Vrai, Faux}\}$ sa fonction de prédicat, on définit :

$$I^+_{\mathcal{P}} = \{i \in I,\ f(i) = \text{Vrai}\}$$

l'ensemble des *instances passives* du problème $\mathcal{P}$ de décision.

Définition 10.2 - *décidabilité d'un problème de décision*

Un problème de décision $\mathcal{P}$ est dit *décidable* lorsqu'il existe $\mathcal{A}$ un algorithme qui pour toute instance du problème $\mathcal{P}$ renvoie la solution attendue.
Autrement dit, pour $f : I \to S$ la fonction de prédicat de $\mathcal{P}$, il existe un algorithme $\mathcal{A}$ tel que :

$$\forall i \in I, f(i) = \mathcal{A}(i)$$

$f(i)$ est ici la solution attendue tandis que $\mathcal{A}(i)$ est la solution attendue pour i.

Le cas échéant, $\mathcal{A}$ termine pour toute instance. *$\mathcal{A}$ résout $\mathcal{P}$* ou que *$\mathcal{P}$ est décidé par $\mathcal{A}$*.

Définition 10.3 - *indécidabilité d'un problème*

Un problème de décision $\mathcal{P}$ est dit *indécidable* lorsqu'il n'existe pas d'algorithme resolvant $\mathcal{P}$.

Remarque 10.4 - *sur l'indécidabilité*

Un problème indécidable est un problème intrinsèquement infaisable : inutile d'essayer de le résoudre car c'est impossible.

Exemple 10.5 - *de problème décidable*

$f : I \to S = \{\text{Vrai, Faux}\}$

$$f(\texttt{l}) = \text{Vrai} \Leftrightarrow \texttt{l} \text{ a exactement 5 éléments}$$

f est la fonction de prédicat d'un problème de décision :

- **Instance** : `l` une liste d'entiers
- **Question** : Est-ce que `l` contient 5 éléments.

Ce problème est décidable car on peut écrire en OCaml :

```
let longueur 5 l =
    match l with
    | e1::e2::e3::e4::e5::[] -> true
    | _ -> false
```

1.1 semi-décidabilité (HP)

Définition 10.6 - *semi-décidabilité d'un problème*

Un problème de décision $\mathcal{P}$ défini par $f : I \to \{\text{Vrai, Faux}\}$ est dit *semi-décidable* lorsqu'il existe un algorithme $\mathcal{A}$ tel que pour $i \in I$:

- si $f(i) = \text{Vrai}$ alors $\mathcal{A}(i) = f(i)$ et $\mathcal{A}$ termine sur i.
- si $f(i) = \text{Faux}$, alors $\mathcal{A}(i) = f(i)$ et $\mathcal{A}$ termine ou bien $\mathcal{A}$ ne termine pas sur i. Ecrire systeme.

Exemple 10.7 - *important*

Il existe des problèmes non semi-décidables. On considère par exemple le suivant :

- **Instance** : une fonction `f : string -> bool` et `f` son code source.
- **Question** : est ce que l'appel `f code_f` ne renvoie pas true ? *i.e.* est ce que l'appel renvoie false ou bien ne termine pas ? N'a-t-on que ces deux possibilités ?

Montrons que ce problème n'est pas semi-décidable.

Supposons par l'absurde qu'il existe un algorithme $\mathcal{A}$ implémenté par une fonction `diag : string -> bool` qui semi-décide le problème. Par définition :

1. `diag code_f` renvoie `true` lorsque `f code_f` ne renvoie pas `true`
2. `diag code_f` renvoie `false` ou ne termine pas lorsque `f code_f` revoie true.

On applique la fonction `diag` à son propre code, noté `code_diag`.

- Si `diag code_diag` renvoie `false` : par définition de `diag` (2.), on a `diag code_diag` renvoie `true`. Il y a alors contradiction.
- Si `diag code_diag` renvoie `true`. Par définition de `diag` (1.), on a `diag code_diag` ne renvoie pas `true`. C'est également absurde.
- Si `diag code_diag` ne termine pas ou échoue, par définition de `diag` (2.), `diag code_diag` renvoie `true`. On a une contradiction.

Aucune possibilité n'est viable. D'où l'absurdité de l'hypothèse.

1.2 Problème de l'arrêt (au programme)

Définition 10.8 - *problème de l'arrêt*

Le *problème de l'arrêt* est le problème qui consiste à décider si un programme ou un algorithme termine sur une entrée.

- **Instance** : un programme `p` donné par son code `code_p` et une entrée `x`
- **Question** : est ce que l'appel `p x` termine ?

Théorème 10.9 - *indécidabilité du problème de l'arrêt*

Le problème de l'arrêt est indécidable.

La démonstration est la suivante. Elle est à connaître impérativement.

Démonstration

Supposons par l'absurde qu'il existe $\mathcal{A}$ un algorithme implément par une fonction `arret` qui résout le problème de l'arrêt. Par définition de décidabilité :

1. `arret code_p x` renvoie `true` lorsque `p x` termine.
2. `arret code_p code_x` renvoie `false` lorsque `p x` ne termine pas.

On écrit :

```
let rec boucle (b:bool) :int =
    match b with
    | true -> boucle true
    | false -> 0

let absurde code_p = boucle (arret code_p code_p)
```

On s'intéresse à l'appel `arret code_absurde code_absurde`.

- Si `arret code_absurde code_absurde` renvoie `true`, par définition de `arret` (1.), `absurde code_absurde` termine. Or cet appel `absurde code_absurde` correspond à `boucle (arret code_absurde code_absurde)` qui ne termine pas par construction, alors que `arret code_absurde code_absurde` renvoie `true` : ceci constitue une contradiction.
- Si `arret code_absurde code_absurde` renvoie `false`. Par définition de `arret` (2.), `absurde code_absurde` ne termine pas. Or `absurde code_absurde` correspond à `boucle (arret code_absurde code_absurde)` qui termine par construction, alors que *arret code_absurde code_absurde* renvoie `false`. Contradiction.

Remarque 10.10 - *usage de chaînes de caractères*

Dans les démonstrations, on préfèrera la manipulation de chaînes de caractères associées à ce qui n'en est pas. En effet, en machine, tout est représenté par des chaînes et en particulier les machines de Turing ne manipulent que ça. C'est plus formel.