Théorie de la complexité — Feuille de cours intégré n° 4
Décidabilité et indécidabilité

1 Décidabilité et indécidabilité

Cette feuille se concentre sur les notions de **décidabilité** et de **calculabilité**. Informellement, un problème de décision est *décidable* s'il existe un algorithme qui le résout (et donc en particulier, qui se termine sur chaque entrée en donnant un verdict oui/non). De même, une fonction est calculable s'il existe un algorithme qui calcule sa sortie en fonction de l'entrée. On rappelle que formellement, un algorithme se modélise par une machine de Turing **totale**. On commence par les fonctions calculables, même si cette notion est plus générale que celle de problème décidable (la raison est qu'on se concentrera plus longuement ensuite sur la notion de problème décidable).

1.1 Fonctions calculables

On rappelle quelques définitions importantes, en commençant par celle des fonctions calculables.

Definition 1 - Fonctions calculables.

Étant donné un alphabet Σ, une fonction totale $f : \Sigma^* \to \Sigma^*$ est dite **calculable** si elle peut s'implémenter par une machine de Turing. Cela signifie qu'il existe une machine de Turing M_f qui **s'arrête sur toute entrée**, et telle que le calcul de la machine M_f sur une entrée x produit $f(x)$ sur la bande lorsque M_f atteint son état d'arrêt.

Remarque 1 - Attention à la phase implicite de codage. On utilisera souvent le terme « calculable » pour désigner des fonctions dont les ensembles de départ et d'arrivée ne sont pas un ensemble de mots Σ^*. Dans ce cas, on suppose avoir implicitement fixé un *codage* des objets considérés par des mots. Par exemple, on peut parler de fonctions $g : \mathbb{N} \to \mathbb{N}$ calculables (ou non), puisqu'on peut coder les entiers en binaire sur l'alphabet $\Sigma = \{0,1\}$. Dans ce cas, l'énoncé « g est calculable » veut dire qu'il existe une fonction $f : \Sigma^* \to \Sigma^*$ qui est calculable (au sens de la Définition 1) telle que pour tout mot $w \in \Sigma^*$, si $n \in \mathbb{N}$ est l'entier codé par w, alors $f(w)$ code $g(n)$. Bien sûr, on peut coder des objets bien plus compliqués que des entiers (par exemple, des machines de Turing, des graphes, etc.).
Attention, dans les énoncés, le codage sera souvent implicite : on parle directement de fonction calculable même si les ensembles de départ et d'arrivée ne sont pas un même ensemble de mots Σ^*.

Exercice 1 – ☆ – Montrer que toutes les fonctions $f : \mathbb{N} \to \mathbb{N}$ décroissantes (au sens large) sont calculables.

Exercice 2 – ★ – Let but est de construire une fonction $f : \mathbb{N} \to \mathbb{N}$ qui est croissante et non-calculable. On utilise la notion de dénombrabilité : on dit qu'un ensemble infini est *dénombrable* s'il est en bijection avec $\mathbb{N}$.

1. Montrer que l'ensemble des fonctions totales **calculables** de $\mathbb{N}$ dans $\{0,1\}$ est dénombrable.
2. Montrer que l'ensemble des fonctions totales de $\mathbb{N}$ dans $\{0,1\}$ n'est pas dénombrable.
3. Justifier qu'il existe une fonction $g : \mathbb{N} \to \{0,1\}$ qui est non-calculable.
4. Utiliser g pour construire la fonction désirée f.

Remarque 2. Comme on l'a vu, une fonction est calculable s'il **existe** une machine de Turing qui l'implémente. Déterminer cette machine peut être difficile, mais c'est une **autre question** que de simplement prouver son existence. Cela est illustré dans l'exercice suivant : n'essayez **pas** de déterminer précisément une machine de Turing pour montrer qu'une fonction est calculable : il suffit de démontrer qu'une telle machine **existe**.

Exercice 3 – ★★☆ – Parmi les cinq fonctions suivantes, quatre sont calculables et savoir si la cinquième est calculable ou non est un problème ouvert. Trouver les quatre fonctions calculables (justifier) :

1. $f : \mathbb{N} \to \{0,1\}$
$$n \mapsto \begin{cases} 1 \text{ si le développement décimal de } \pi \text{ contient un nombre fini de 1} \\ 0 \text{ sinon} \end{cases}$$

2. $f : \mathbb{N} \to \{0,1\}$
$$n \mapsto \begin{cases} 1 \text{ si } \pi \text{ contient exactement } n \text{ fois le chiffre 1 dans son développement décimal} \\ 0 \text{ sinon} \end{cases}$$

3. $f : \mathbb{N} \to \{0,1\}$
$$n \mapsto \begin{cases} 1 \text{ si } \pi \text{ contient au moins } n \text{ fois le chiffre 1 dans son développement décimal} \\ 0 \text{ sinon} \end{cases}$$

4. $f : \mathbb{N} \to \{0,1\}$
$$n \mapsto \begin{cases} 1 \text{ si } \pi \text{ contient un bloc } \textit{maximal} \text{ de exactement } n \text{ chiffres 1 consécutifs} \\ \quad \text{dans son développement décimal} \\ 0 \text{ sinon} \end{cases}$$

5. $f : \mathbb{N} \to \{0,1\}$
$$n \mapsto \begin{cases} 1 \text{ si } \pi \text{ contient un bloc } \textit{maximal} \text{ de au moins } n \text{ chiffres 1 consécutifs} \\ \quad \text{dans son développement décimal} \\ 0 \text{ sinon} \end{cases}$$

1.2 Décidabilité, indécidabilité et semi-décidabilité

On passe maintenant aux langages décidables et indécidables. Cette définition considère les machines de Turing **de décision**.

Definition 2 - Langages décidables.

Soit Σ un alphabet et $L \subseteq \Sigma^*$ un langage. On dit que L est *décidable* (ou récursif), quand il existe une machine de Turing (de décision) M telle que pour tout mot $w \in \Sigma^*$:

- $w \in L$ si et seulement si M accepte w.
- $w \notin L$ si et seulement si M rejette w.

Symétriquement, un langage qui n'est pas décidable est dit *indécidable*.

Observons que quand un langage est décidable, une machine de Turing qui témoigne de cette propriété **doit s'arrêter sur toute entrée**. On va affaiblir cette condition pour obtenir une autre notion : la semi-décidabilité.

Definition 3 - Langages semi-décidables.

Soit Σ un alphabet et $L \subseteq \Sigma^*$ un langage. On dit que L est *semi-décidable* quand il existe une machine de Turing (de décision) M telle que pour tout mot $w \in \Sigma^*$, $w \in L$ si et seulement si M accepte w.

On peut reformuler la notion de langage semi-décidable. Soit M une machine de Turing. Le langage accepté par M est l'ensemble $\mathscr{L}(M)$ des mots qu'elle accepte. Ainsi,

$$\mathscr{L}(M) = \{ w \in \Sigma^* \mid M \text{ accepte } w \}.$$

Un langage L est donc semi-décidable quand il existe une machine M telle que $L = \mathscr{L}(M)$.

On remarque que la définition des langages décidables est plus contraignante que celle des langages semi-décidables. Si L est semi-décidable, une machine qui témoigne de cette propriété a deux comportements possibles sur un mot d'entrée $w \notin L$: soit elle rejette, soit elle ne termine pas. Ainsi, tout langage décidable est en particulier semi-décidable. On résume la situation dans la Figure 1 (lire attentivement la légende).

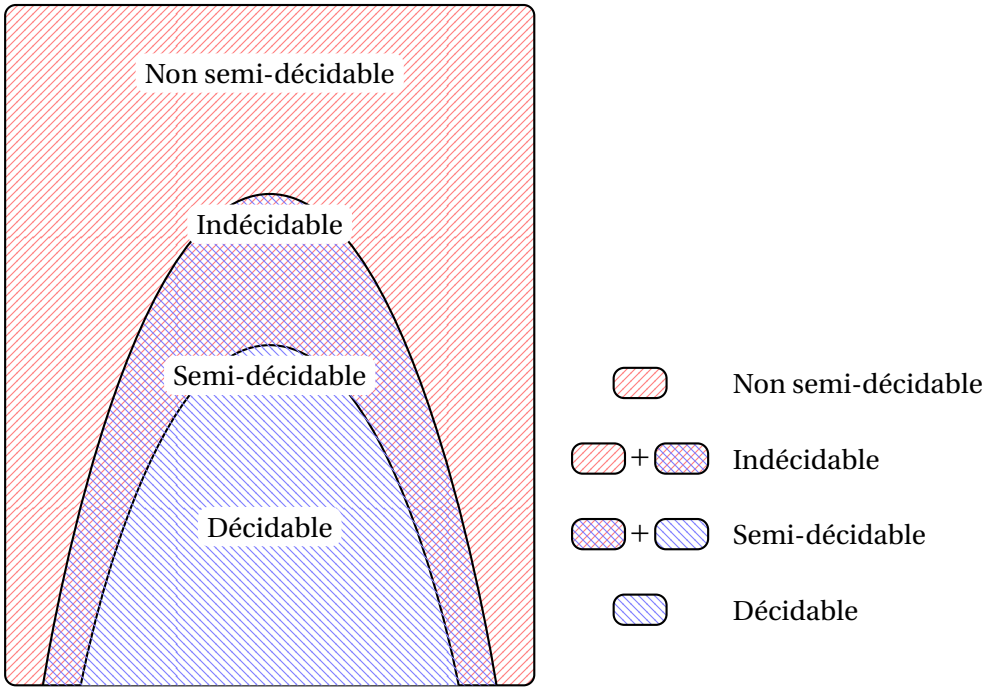

FIGURE 1 – Comparaisons des classes de langages

Remarque 3 - Des langages aux problèmes de décision : encore et toujours le codage. De même que pour le terme « calculable », on utilisera souvent les termes « décidable », « indécidable » et « semi-décidable » pour désigner des ensembles qui de ne sont pas des langages (*i.e.*, contenant autre chose que des mots). Encore une fois, cela suppose qu'un codage est fixé. Par exemple, on dira que l'ensemble $E \subseteq \mathbb{N}$ des entiers pairs est décidable car le langage des mots de $\{0,1\}^*$ codant un entier pair est décidable (au sens de la Définition 2). De même, l'ensemble des graphes complets (toutes les paires de sommets forment une arête) est décidable car on peut coder un graphe par un mot de $\{0,1\}^*$ et le langage des mots codant un graphe complet est décidable (au sens de la Définition 2). Comme expliqué ci-dessus, le codage est souvent implicite dans les énoncés.

Enfin, dans ce contexte, il est usuel (et équivalent) de parler de « problèmes de décision » au lieu d'ensembles. Par exemple, les deux ensembles ci-dessus correspondent aux problèmes suivants :

Parité

ENTRÉE : Un entier $n \in \mathbb{N}$.
QUESTION : Est-ce que n est pair?

Complétude

ENTRÉE : Un graphe G.
QUESTION : Est-ce que G est complet?

Nous sommes maintenant prêts à donner un premier exemple de problème indécidable : le problème **DIAG**, qui consiste à déterminer si une machine de Turing n'accepte pas son propre code.

Remarque 4 - Des machines de Turing en entrée. Avant d'énoncer et de prouver ce théorème, une remarque importante s'impose : l'**entrée** du problème **DIAG** est une machine du Turing (et la question posée porte sur cette machine). Pour se donner une machine de Turing en entrée, il faut donc représenter cette machine par un mot, c'est-à-dire coder par un unique mot ses états, ses alphabets (d'entrée et de travail) et ses instructions. Cette information est finie, elle peut donc se représenter comme un mot fini, par exemple sur l'alphabet $\{0,1\}$. Il s'agit donc encore une fois d'une question de codage.
Même si ce n'est pas intéressant en soi, on peut expliciter un tel codage pour se convaincre qu'on peut le faire facilement. Quitte à les renommer, on peut par exemple représenter les états par des entiers $1,\dots,n$, où 1 est l'état initial, et n l'état final (ou bien, dans le cas d'une machine de décision, où $n-1$ est l'état q_r et n est l'état q_a). Si l'alphabet de travail a p lettres, *i.e.*, $\Gamma=\{a_1,\dots,a_p\}$, chaque transition est alors de la forme,

$$\delta(i,a_j)=(k,a_\ell,m),$$

où i,k sont des états et où $m\in\{1,2,3\}$ code le mouvement de la tête de lecture-écriture de la machine (par exemple : 1 signifie *gauche*, 2 signifie *droite* et 3 signifie *rester sur place*). On peut coder une telle transition par le mot binaire

$$1^i01^j01^k01^\ell01^m,$$

(où 1^x représente le mot $\underbrace{11\cdots1}_{x \text{ fois}}$).
Par exemple avec ce codage, la transition $\delta(3,a_2)=(5,a_1,1)$ se représenterait par le mot :

$$1110110111110101.$$

Dans ce codage, le symbole 0 est ainsi utilisé pour séparer les différents blocs de 1 les uns des autres, ce qui permet de retrouver, à partir d'une telle chaîne de 0 et de 1, les entiers i,j,k,ℓ,m dont elle provient, et donc de la décoder de façon unique.
Il faut aussi noter qu'il n'y a aucun bloc de deux 0 consécutifs dans un tel codage (parce que i,j,k,ℓ,m valent tous au moins 1). Si on note code(t) le code d'une transition t défini plus haut, on peut coder une suite d'instructions $t_1,t_2,\dots,t_r$ par le mot binaire,

$$\text{code}(t_1)00\text{code}(t_2)00\dots00\text{code}(t_{r-1})00\text{code}(t_r).$$

Autrement dit, on sépare les codes des instructions par des blocs 00. Comme le code d'une unique transition ne contient pas de bloc 00, il est très simple de décoder une telle chaîne de façon unique. On a donc proposé une représentation des machines de Turing par mots binaires.
En pratique, on ne s'intéresse pas à la façon dont sont codées les machines de Turing. Il faut juste retenir qu'on peut fixer un codage. Une fois un codage fixé, on notera le code d'une machine M par $c(M)$, ou $[M]$ ou encore $\langle M\rangle$.

Une fois fixé un codage des machines de Turing, on peut formuler le problème **DIAG** ainsi :

Problème **DIAG**.

- **ENTRÉE** : Une machine de Turing M, donnée par son code $c(M)$.
- **QUESTION** : Est-il vrai que la machine M n'accepte ***pas*** son propre code $c(M)$?

De façon équivalente, on peut voir **DIAG** comme un langage, celui des codes de machines de Turing qui n'acceptent pas leur propre code :

$$\begin{aligned}\textbf{DIAG}&=\left\{\,c(M)\;\middle|\;M \text{ n'accepte pas l'entrée } c(M)\,\right\}\\&=\left\{\,c(M)\;\middle|\;c(M)\notin\mathscr{L}(M)\,\right\}.\end{aligned}$$

Le résultat suivant énonce que le problème **DIAG** est indécidable. La preuve est simple : il suffit de raisonner par l'absurde.

Théorème 5

Le problème **DIAG** est indécidable.

Proof

Supposons par l'absurde que **DIAG** soit décidable. Il existerait alors une machine de Turing M_0 qui s'arrête toujours telle que pour toute machine de Turing M :

$$\text{La machine } M_0 \text{ accepte } c(M) \text{ si et seulement si } c(M) \text{ appartient à } \textbf{DIAG}.$$

Par définition de **DIAG**, cette machine M_0 aurait donc la propriété suivante :

$$\text{La machine } M_0 \text{ accepte } c(M) \text{ si et seulement si } M \text{ n'accepte pas } c(M).$$

En particulier, cette équivalence doit être vraie pour $M = M_0$. Donc, M_0 accepte $c(M_0)$ si et seulement si M_0 n'accepte pas $c(M_0)$. C'est une contradiction. L'hypothèse faite au début de cette démonstration est donc fausse. Donc **DIAG** est indécidable.

D'autres problèmes indécidables

Maintenant que nous avons en main un problème indécidable, nous pouvons l'utiliser pour montrer que d'autres problèmes sont aussi indécidables. On va s'intéresser au problème **HALT** suivant :

Problème **HALT**.

- **ENTRÉE** : Une machine de Turing M donnée par son code $c(M)$ et un mot w écrit sur l'alphabet d'entrée de M.
- **QUESTION** : Est-ce que l'exécution de M sur w termine ?

Autrement dit,

$$\textbf{HALT} = \left\{ \left(c(M), w\right) \mid M \text{ s'arrête sur } w \right\}.$$

Notre objectif est de prouver le théorème suivant.

Théorème 6 — **Indécidabilité du problème de l'arrêt**

Le problème **HALT** est semi-décidable et indécidable.

Avant de prouver ce théorème, faisons une remarque importante, à lire avant et après la lecture de la preuve.

Remarque 7 - Une preuve générique. Ce qui est important dans la preuve du Théorème 6 est qu'elle met en œuvre un raisonnement *générique*, qu'on utilisera pour montrer l'indécidabilité de nombreux autres problèmes. Ce raisonnement générique fonctionne de la façon suivante : on choisit d'abord un problème que l'on sait déjà indécidable. Pour l'instant, nous n'avons pas le choix, puisque nous connaissons un seul problème indécidable : **DIAG**. On montre ensuite que si **HALT** était décidable, on pourrait utiliser la machine de Turing supposée décider **HALT** pour construire une machine de Turing qui décide **DIAG**. Comme **DIAG** est indécidable, une telle machine n'existe donc pas. La conclusion est que **HALT** est également indécidable. Cette technique générique se formalise grâce à un outil mathématique simple appelé ***réduction***. La Section 2 est consacrée aux réductions, une notion centrale de ce cours.

Démonstration du Théorème 6

Le fait que **HALT** est semi-décidable est facile : on considère une machine de Turing prenant en entrée $c(M)$ et w qui simule M sur w, et accepte si cette simulation s'arrête. Par définition, cette machine accepte le couple $(c(M), w)$ si et seulement si le calcul de M s'arrête sur w, c'est-à-dire si et seulement si $(c(M), w) \in \textbf{HALT}$.

Pour montrer que le problème est indécidable, on raisonne par l'absurde. Supposons que **HALT** soit décidable : il existerait une machine de Turing $M_{\textbf{HALT}}$ qui prend en entrée une machine M et un mot w, et

- accepte si le calcul de M s'arrête sur w,
- rejette si le calcul de M ne s'arrête pas sur w.

À partir de $M_{\textbf{HALT}}$, on va construire une nouvelle machine $M_{\textbf{DIAG}}$ prenant en entrée le code $c(M)$ d'une machine M, et qui décide le problème **DIAG**. Si on parvient à construire $M_{\textbf{DIAG}}$, on aura donc une contradiction, puisqu'on sait que **DIAG** est indécidable (Théorème 5), ce qui montrera que l'hypothèse d'existence de $M_{\textbf{HALT}}$ est absurde.

La machine $M_{\textbf{DIAG}}$ doit décider le langage $\textbf{DIAG} = \{ c(M) \mid c(M) \notin \mathscr{L}(M) \}$, c'est-à-dire accepter les (codes des) machines de Turing qui n'acceptent pas leur propre code, et rejeter les autres. Elle fonctionne ainsi :

- $M_{\textbf{DIAG}}$ duplique son entrée $c(M)$ pour obtenir un couple $(c(M), c(M))$.
- Elle lance la machine $M_{\textbf{HALT}}$ sur l'entrée $(c(M), c(M))$. Comme $M_{\textbf{HALT}}$ est supposée décider le problème **HALT**, cette machine s'arrête toujours.
  - Si $M_{\textbf{HALT}}$ rejette, cela signifie, par définition de $M_{\textbf{HALT}}$, que M ne s'arrête pas sur son propre code $w = c(M)$. Dans ce cas, $c(M) \notin \mathscr{L}(M)$, et $M_{\textbf{DIAG}}$ accepte.
  - Si $M_{\textbf{HALT}}$ accepte, cela signifie, par définition de $M_{\textbf{HALT}}$, que M s'arrête sur son propre code $c(M)$. Dans ce cas, la machine $M_{\textbf{DIAG}}$ simule M sur son propre code. On sait que cette simulation s'arrête. Si elle s'arrête sur l'état acceptant de M, c'est que M accepte son propre code ($c(M) \in \mathscr{L}(M)$), et la machine $M_{\textbf{DIAG}}$ rejette. Sinon, c'est que M rejette son propre code, et en particulier, elle ne l'accepte pas ($c(M) \notin \mathscr{L}(M)$). Dans ce cas, la machine $M_{\textbf{DIAG}}$ accepte.

On a montré que s'il existait une machine de Turing $M_{\textbf{HALT}}$ décidant **HALT**, il existerait une machine de Turing $M_{\textbf{DIAG}}$ décidant **DIAG**. Comme une telle machine n'existe pas (Théorème 5), il n'existe pas de machine de Turing décidant **HALT**, qui est donc un problème indécidable.

Remarque 8 - Retour sur la preuve d'indécidabilité de HALT. La preuve précédente fonctionne de la façon suivante : à partir d'une entrée $c(M)$ de **DIAG**, elle explique comment construire une entrée $(c(M'), w)$ de **HALT** telle que,

$$c(M) \in \textbf{DIAG} \iff (c(M'), w) \in \textbf{HALT}.$$

De plus, cette construction est algorithmique, c'est-à-dire, implémentable par machine de Turing (dans notre cas, $M' = M$ et $w = c(M)$). La fonction calculable $c(M) \mapsto (c(M'), w)$ s'appelle une **réduction** du problème **DIAG** au problème **HALT**. Cette notion sera formalisée ultérieurement.

Exercice 4 – ☆ – Soit Σ un alphabet et $L \subseteq \Sigma^*$. On suppose que L et $\Sigma^* \setminus L$ sont tous deux semi-décidables (on dit que L est à la fois semi-décidable et co-semi-décidable). Montrer que L est décidable.

Exercice 5 – ☆ – On considère le problème **un-HALT** suivant :

- **ENTRÉE** : Une machine de Turing M et un mot w écrit sur l'alphabet d'entrée de M.
- **QUESTION** : Est-ce que l'exécution de M sur w ne termine pas ?

Le problème **un-HALT** est-il décidable ? Est-il semi-décidable ?

Exercice 6 – ★ – Soit $X \subseteq \mathbb{N}$ *semi-décidable*. Montrer qu'il existe un ensemble *décidable* $Z \subseteq \mathbb{N} \times \mathbb{N}$ tel que :

$$X = \{ x \mid \exists y \text{ tel que } (x, y) \in Z \}.$$

Exercice 7 – ★★☆ – Soit l'alphabet $\Sigma = \{0, 1\}$ et deux langages $K, L \subseteq \Sigma^*$ semi-décidables (potentiellement non disjoints, *i.e.*, $K \cap L \neq \emptyset$). Construire deux autres langages semi-décidables $K', L' \subseteq \Sigma^*$ tels que :

- $K' \subseteq K$,
- $L' \subseteq L$,
- $K' \cap L' = \emptyset$,
- $K' \cup L' = K \cup L$.

Exercice 8 – ❗★ – On considère l'alphabet $\Sigma = \{0,1\}$. Étant donné un langage $L \subseteq \Sigma^*$, on dit qu'*une machine M énumère L* si l'exécution de M sur le mot vide (ε) écrit successivement tous les mots de L (et uniquement ceux-ci) sur sa bande. En particulier, si L est infini, l'exécution de M ne termine pas (on sait juste que tout mot de L finira un jour par être écrit sur la bande de M).
Étant donné un langage $L \subseteq \Sigma^*$ montrer que L est semi-décidable si et seulement si il existe une machine de Turing qui énumère L (d'où le terme récursivement énumérable).

2 Réductions — Comment montrer l'indécidabilité d'un problème

Maintenant que nous connaissons un premier problème indécidable (**HALT**), nous allons utiliser une nouvelle méthode pour prouver que d'autres problèmes indécidables existent : les *réductions*.

Definition - Réductions.

Considérons un alphabet Σ et deux langages $K, L \subseteq \Sigma^*$. Une réduction de K vers L est une fonction **calculable** $f : \Sigma^* \to \Sigma^*$ telle que :

$$\text{Pour tout } w \in \Sigma^*, \quad w \in K \text{ si et seulement si } f(w) \in L$$

On peut maintenant énoncer le théorème que nous utiliserons pour prouver que des langages (et les problèmes qu'ils encodent) sont indécidables.

Théorème 9

Soit Σ un alphabet et deux langages $K, L \subseteq \Sigma^*$ tels qu'il existe une réduction de K vers L. Alors, on a :

1. Si L est décidable alors K est décidable.
2. Si K est indécidable alors L est indécidable.
3. Si L est semi-décidable alors K est semi-décidable.
4. Si K n'est pas semi-décidable alors L n'est pas semi-décidable.

Le Théorème 9 donne une nouvelle méthode pour prouver qu'un langage est indécidable. On dispose d'un « stock » de problèmes indécidables (qui pour l'instant n'en contient que deux : **DIAG** et **HALT**). Si on veut prouver qu'un nouveau problème P est indécidable, on peut choisir (astucieusement) un problème Q déjà dans le stock et exhiber une réduction de Q vers P. On obtient ensuite l'indécidabilité de P par la seconde propriété du Théorème 9.

Exercice 9 – ❗★ – Prouver le Théorème 9.

Exercice 10 – ❗★ – Montrer que la relation de réduction entre langages est transitive, c'est-à-dire si L_1 se réduit à L_2 et L_2 se réduit à L_3, alors L_1 se réduit à L_3.

Exercice 11 – ❗★★ – Parmi les problèmes suivants, lesquels sont décidables, lesquels sont semi-décidables et lesquels sont indécidables ? Justifier chaque affirmation (en particulier l'indécidabilité doit être prouvée par une réduction).

- **HALT**$_\varepsilon$: étant donnée une machine de Turing M, est-ce que M s'arrête sur le mot vide "ε" ?
- **UNIV** : étant donnée une machine de Turing de décision M et un mot d'entrée w, est-ce que M accepte w ?
- **HALT**$_{\textbf{timed}}$: étant donnée une machine de Turing M, un mot d'entrée w et un entier t, est-ce que M accepte w en moins de t étapes ?
- **HALT**$_{\textbf{or}}$: étant données deux machines de Turing M_1, M_2 et un mot d'entrée w, est-ce que M_1 ou M_2 s'arrête sur w ?

- **HALT$_\exists$** : étant donnée une machine de Turing M, est-ce qu'il existe un mot w tel que M s'arrête sur w ?
- **STATE** : étant donnée une machine de Turing M et un entier n, est-ce que M a n états ?
- **HALT$_\forall$** : étant donnée une machine de Turing M, est-ce que M s'arrête sur tous les mots d'entrée w ?
- **EQUIV** : étant données deux machines de Turing M_1, M_2 est-ce que M_1 et M_2 donnent le même résultat pour tout mot d'entrée w ?
- **PI** : étant donnée une machine de Turing M, est-ce que M calcule π ? C'est-à-dire que pour tout $i \in \mathbb{N}$, M s'arrête sur l'entrée i et écrit le i-ème chiffre du développement décimal de π.

3 Théorème de Rice

On présente maintenant le théorème de Rice qui généralise une partie de l'exercice 11.

Théorème 10

On considère un alphabet Σ. Soit F l'ensemble de toutes les fonctions partielles de Σ^* dans Σ^* implémentées par une machine de Turing. On considère un sous-ensemble $E \subseteq F$ tels que $E \neq \emptyset$ et $E \neq F$. Le problème suivant est indécidable :

- **ENTRÉE** : Une machine de Turing M.
- **QUESTION** : Est-ce que la fonction partielle implémentée par M appartient à E ?

Une façon moins formelle d'énoncer le théorème de Rice est la suivante : « toute propriété **sémantique** et non-triviale des machines de Turing est indécidable ».

- Le terme « sémantique » veut dire la propriété ne doit dépendre que de la sémantique de la machine (*i.e.*, de la fonction partielle qu'elle implémente).
- Le terme « non-triviale » veut dire qu'il existe au moins une machine qui la satisfait et une autre qui ne la satisfait pas (celà correspond à l'hypothèse que $E \neq \emptyset$ et $E \neq F$ dans le théorème).

Exercice 12 – ★ – Montrer que pour tous les problèmes indécidables dans l'exercice 11, leur indécidabilité est une conséquence immédiate du théorème de Rice.

Exercice 13 – ★★ – On considère la sous-classe des machines de Turing *linéairement bornées*. Sur un mot d'entrée de taille $n \in \mathbb{N}$, une telle machine utilise un espace borné par n.

- Montrer que **HALT** est décidable pour les machines linéairement bornées : il existe une machine de Turing qui termine et répond correctement au problème de l'arrêt quand on lui passe en entrée une machine linéairement bornée (on ignore ce que la machine fait pour les autres entrées).
- Montrer que le problème **HALT$_\exists$** restreint aux machines linéairement bornées reste indécidable.

Exercice 14 – ★★ – Parmi les problèmes suivants, lesquels sont décidables ?

- Étant donné une machine de Turing M sur l'alphabet d'entrée $\Sigma = \{0,1\}$, est-il vrai que pendant l'exécution de M sur le mot vide "ε", les transitions déplacent toujours la tête de lecture vers la droite ?
- Étant donné une machine de Turing M sur l'alphabet d'entrée $\Sigma = \{0,1\}$, est-il vrai que pendant l'exécution de M sur le mot vide "ε", une transition écrit une lettre de Σ sur la bande ?
- Étant donné une machine de Turing M sur l'alphabet d'entrée $\Sigma = \{0,1\}$, est-il vrai que pendant l'exécution de M sur le mot vide "ε", une transition écrit la lettre de "1" sur la bande ?