Calculabilité et Complexité (11X008) - Printemps 2026

3. Réductions entre langages

Enseignant: Arnaud Casteigts

Assistants: M. De Francesco, M. Marseloo
Moniteurs: E. Bussod, N. Beghdadi

Comme nous savons, Turing a montré en 1936 qu'il existe des langages indécidables, notamment, le langage de l'arrêt $L_H = \{(\langle M \rangle, w) \mid M \text{ termine sur l'entrée } w\}$.

La bonne nouvelle est qu'une fois que l'on sait que certains langages sont indécidable, on peut les utiliser pour montrer que d'autres le sont aussi, en utilisant le concept de **réduction entre langages**. Dans ce cours, nous présentons ce concept et nous donnons quelques exemples, notamment un exemple simple et un exemple plus compliqué. Puis nous mentionnons d'autres réductions de ce type sans les prouver.

3.1 Qu'est-ce qu'une réduction ?

Supposons que l'on sache déjà qu'un langage L_1 est indécidable et que l'on veuille montrer qu'un autre langage L_2 est indécidable. On peut commencer par supposer l'existence d'une machine M_2 pour décider L_2. Puis, voir si une telle machine peut être utilisée pour créer une machine M_1 qui décide L_1. Si c'est le cas, la décidabilité de L_2 vient alors contredire l'indécidabilité de L_1, ce qui implique que L_2 est lui-aussi indécidable. (Lisez deux fois si besoin, c'est important.)

Définition 3.1 (Réduction)**.** Étant donnés deux langages L_1 et L_2, L_1 *se réduit* à L_2 si l'existence d'une machine décidant L_2 implique l'existence d'une machine décidant L_1. On écrit cela $L_1 \leq_T L_2$ (où T signifie Turing).

Quand vous programmez, vous faites cela intuitivement. Par exemple si vous écrivez un programme qui résoud A en utilisant une fonction qui résoud B, alors vous réduisez A à B.

3.1.1 Exemple simple

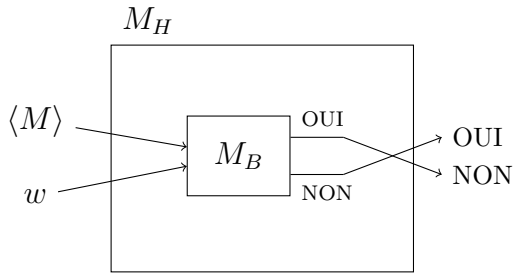
Soit le langage $L_B = \{(\langle M \rangle, w) \mid M \text{ boucle à l'infini sur l'entrée } w\}$.

Nous voulons montrer que L_B est indécidable. Nous allons le faire en montrant que L_H se réduit à L_B.

Lemme 3.2. *L_H se réduit à L_B.*

Preuve : Supposons qu'il existe une machine M_B qui décide L_B. On peut alors créer une machine M_H qui décide L_H en passant la même entrée à M_B et en inversant la réponse. Le pseudo-code correspondant est indiqué à gauche (et un diagramme équivalent à droite).

```
def M_H(⟨M⟩, w) :
    if M_B(⟨M⟩, w) accepts :
        reject
    else :
        accept
```



□

Nous savions déjà que L_H est indécidable, mais nous avons montré qu'on peut le décider si L_B est décidable (contradiction). Donc L_B est indécidable.

Vous avez peut-être remarqué que L_B n'est autre que le complément de L_H, donc nous aurions pu conclure directement en utilisant le fait plus général que le complément d'un langage indécidable est toujours indécidable (pourquoi ?). Cependant, ignorer cela nous a permis d'illustrer le concept de réduction de manière très simple.

3.1.2 Exemple plus compliqué

Soit le langage $L = \{(\langle M \rangle, w) \mid M \text{ accepte } w\}$.

Lemme 3.3. *L_H se réduit à L.*

Preuve. Comme à chaque fois, le but est d'utiliser une hypothétique machine M_L qui décide L afin de créer une machine M_H qui décide L_H.

Notre construction va utiliser une troisième machine M_S, dont le rôle est assez simple. Étant donnée une machine $\langle M \rangle$ et son entrée w, le rôle de M_S est de simuler $M(w)$ et d'attendre que la simulation termine, puis terminer en acceptant. Clairement, la machine M_S existe, ce n'est qu'une machine universelle un peu modifiée, donc la description $\langle M_S \rangle$ d'une telle machine existe aussi. C'est elle que nous allons utiliser.

La propriété clé est la suivante : $M_S(\langle M \rangle, w)$ accepte si et seulement si $M(w)$ termine. Si l'on veut savoir si $M(w)$ termine, il suffit donc de savoir si $M_S(\langle M \rangle, w)$ accepte. Cela tombe bien, c'est exactement ce que l'hypothétique machine M_L est censée pouvoir décider.

Toutes les pièces du puzzle sont maintenant disponible pour créer la machine M_H qui résoud le problème de l'arrêt. Pour rappel, cette machine M_H doit décider, pour une machine $\langle M \rangle$ et une entrée w données, si $M(w)$ termine ou pas. Il nous suffit alors d'exécuter

$M_L(\langle M_S \rangle, (\langle M \rangle, w))$. Soit cette exécution accepte, soit elle rejette. Or, elle accepte si et seulement si $M_S(\langle M \rangle, w)$ accepte, et donc si et seulement si $M(w)$ termine (propriété clé ci-dessus). On a donc bien résolu le problème de l'arrêt !

Cette construction est illustrée à la Figure 1.

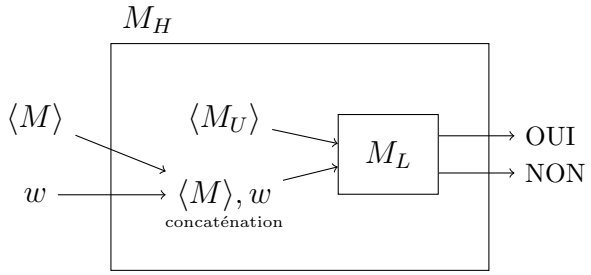


FIGURE 1 – Machine qui décide L_H à l'aide d'une machine qui décide L.

□

Comme tout à l'heure, nous pouvons maintenant conclure : Nous savions déjà que L_H est indécidable, mais nous avons montré qu'on peut le décider si L est décidable (contradiction). L est donc indécidable.

Pour information, le langage L est appelé *langage universel* et souvent noté L_U. Nous n'avons pas donné son nom plus haut pour éviter les confusions avec la machine universelle M_U, qui reconnaît bel et bien L_U (quand on y pense), mais qui ne le décide pas.

3.2 Autres langages indécidables

3.2.1 Langage vide

Étant donné une machine $\langle M \rangle$, on souhaite décider si cette machine n'accepte aucun mot, autrement dit, décider si son langage est vide : $L_\emptyset = \{\langle M \rangle \mid L(M) = \emptyset\}$.

Nous allons montrer que $L_\emptyset$ est indécidable. Pour cela, nous allons réduire L_U à $L_\emptyset$. Et comme L_U est indécidable, cela montre que $L_\emptyset$ l'est aussi.

Réduction : Supposons qu'il existe une machine $M_\emptyset$ qui décide $L_\emptyset$. Nous allons utiliser $M_\emptyset$ pour construire une machine M_U qui *décide* L_U (cette machine sera différente de la machine M_U habituelle, qui ne fait que reconnaître L_U).

Nous donnons ici la réduction sous forme d'un programme en pseudo-code. Ce programme définit une fonction interne et suppose qu'il peut accéder à la description de cette fonction. Cela peut sembler bizarre, mais c'est tout à fait valide (dans le sens où ce type de choses peuvent être réalisées par (et sur) des machines de Turing).

```
def M_U(⟨M⟩, w) :
    def M'(x) :
        if x == w :
            return M(x)
        else :
            reject
    if M_∅(⟨M'⟩) accepts :   //L(M) ≠ ∅
        reject
    else :                   //L(M) = ∅
        accept
```

En entrée, M_U prend une machine $\langle M \rangle$ et une entrée w et on veut décider si M accepte w. On commence par créer une machine intermédiaire M' qui rejette toutes les entrées sauf w, et si l'entrée est w, elle renvoit $M(w)$. Observons qu'on a alors $L(M') \neq \emptyset$ si et seulement si M accepte w. On peut donc utiliser maintenant $M_\emptyset$ pour savoir si $L(M')$ est vide ou non, et conclure selon la réponse que M accepte w ou non.

3.2.2 Le problème du langage non-vide

Étant donné la description de machine $\langle M \rangle$, on souhaite décider si M accepte au moins un mot, autrement dit, on s'intéresse au langage complément de $L_\emptyset$:

$\overline{L_\emptyset} = \{\langle M \rangle \mid L(M) \neq \emptyset\}$.

Le fait que $L_\emptyset$ soit indécidable implique que $\overline{L_\emptyset}$ l'est aussi. (Par l'absurde, si $\overline{L_\emptyset}$ était décidable, alors $L_\emptyset$ serait décidable... souvenez-vous que la famille des langages décidables est stable par complémentation.) Vous verrez en exercice que $\overline{L_\emptyset}$ est cependant reconnaissable, et on peut déduire de cela que $L_\emptyset$ ne peut pas être reconnaissable. (Pourquoi ? Réfléchissez-y par l'absurde à nouveau.)

3.2.3 Problème de correspondance de Post

Le problème de correspondance de Post (PCP) est un problème plus naturel que les précédents, car il ne s'agit pas d'un langage de description de machines. Étant données deux listes de k mots chacune,

$$A = x_1, \ldots, x_k \qquad B = y_1, \ldots, y_k,$$

est-t-il possible de construire un mot w en concaténant certains mots de A, de sorte que la concaténation des mots de B *ayant les mêmes indices* donnent aussi w. Les répétitions sont autorisées.

Par exemple, prenons $A = (\texttt{b}, \texttt{a}, \texttt{ca}, \texttt{abc})$ et $B = (\texttt{ca}, \texttt{ab}, \texttt{a}, \texttt{c})$. On peut trouver une solution qui correspond à la suite d'indices $2, 1, 3, 2, 4$. Cela donne :

`a.b.ca.a.abc` (pour A)

`ab.ca.a.ab.c` (pour B)

En revanche, pour $A = (ba, abb, bab)$ et $B = (bab, bb, abb)$, il n'existe aucune solution.

Le problème de correspondance de Post est celui de décider s'il existe une solution, autrement dit à décider le langage :

$L_P = \{A = x_1, \ldots, x_k; B = y_1, \ldots, y_k \mid \exists i_1, i_2, ..., i_\ell,\ x_{i_1} x_{i_2} ... x_{i_\ell} = y_{i_1} y_{i_2} ... y_{i_\ell}\}$.

Ce problème est indécidable. Nous ne ferons pas la preuve, mais c'est bon à retenir, car de nombreux problèmes ont été montrés indécidable en utilisant une réduction depuis PCP.