Calculabilité et Complexité (11X008) - Printemps 2025

3. Propriétés de clôture et réductions

Enseignant: Arnaud Casteigts

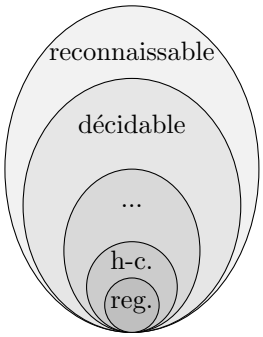
Assistants: A. Berger, M. De Francesco
Moniteurs: E. Bussod, N. Beghdadi

Dans ce cours, nous allons voir comment les familles des langages décidables et des langages reconnaissables se comportent sous les opération d'union, d'intersection et de complément. Puis nous allons présenter la notion de réduction entre problèmes (entre langages) qui sera centrale dans le reste du cours (y compris pour la partie complexité).

3.1 Décidable versus reconnaissable (rappels)

Le fait que les machines de Turing ne terminent pas toujours impose de distinguer deux familles distinctes : les langages décidables et les langages reconnaissables.

Un langage L est **décidable** s'il existe une machine M qui accepte les mots L et rejette les mots de $\overline{L}$. Un langage L est **reconnaissable** s'il existe une machine M qui accepte les mots de L et n'accepte pas les mots de $\overline{L}$ (en rejettant ou en bouclant). Un langage décidable est donc aussi reconnaissable, mais l'inverse n'est pas forcément vrai.



Synonymes :
- décidable : *récursif*
- reconnaissable : *récursivement énumérable*, *semi-décidable*.

Certains langages sont reconnaissables, mais pas décidables. Par exemple, le langage de l'arrêt $L_H = \{(\langle M \rangle, w) \mid M \text{ termine sur l'entrée } w\}$ n'est pas décidable, mais il est reconnaissable : il suffit d'utiliser une machine universelle M_U qui simule M sur l'entrée w et se contente de répondre OUI lorsque la simulation se termine. Si la simulation ne termine pas, M_U ne termine pas non plus, mais ce n'est pas un problème pour *reconnaître* L_H.

3.2 Propriétés de ces langages

Voyons quelques propriétés des langages décidables ou reconnaissables, notamment comment ces familles se comportent sous les opérations d'union, d'intersection et de complément. Pour rappel, $w \in L_1 \cup L_2$ signifie que w est dans L_1 ou dans L_2 (ou les deux) ; $w \in L_1 \cap L_2$ signifie que w est dans L_1 et dans L_2 ; et $w \in \overline{L}$ signifie que w n'est pas dans L.

Commençons par quelques rappels sur les autres familles que nous connaissons.

Langages réguliers :

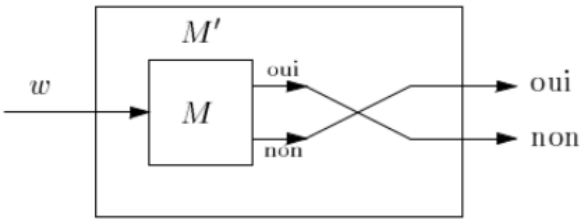
- Si L_1 est régulier et L_2 est régulier, alors $L_1 \cup L_2$ est régulier
- Si L_1 est régulier et L_2 est régulier, alors $L_1 \cap L_2$ est régulier
- Si L est régulier, alors $\overline{L}$ est régulier

Langages hors-contextes :

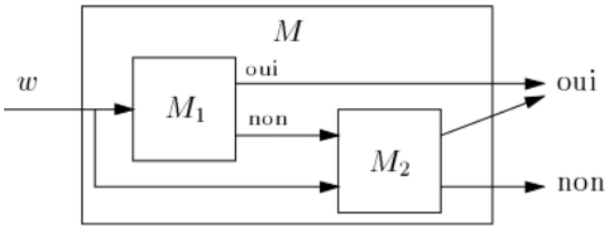
- Si L_1 et L_2 sont hors-contextes, alors $L_1 \cup L_2$ est hors-contexte.
- Si L_1 et L_2 sont hors-contextes, alors $L_1 \cap L_2$ n'est pas forcément hors-contexte.
- Si L est hors-contexte, alors $\overline{L}$ peut ne pas être hors-contexte.

3.2.1 Langages décidables

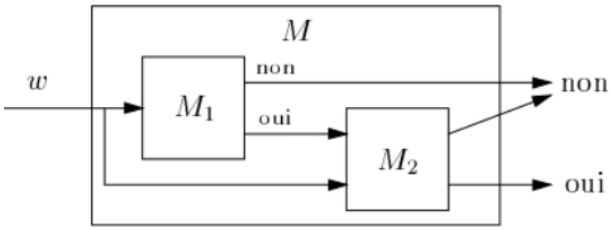
- Si L est décidable, alors $\overline{L}$ est décidable.
  Preuve : Soit L un langage décidable. Il existe donc une machine de Turing M qui s'arrête toujours et décide L. Pour décider $\overline{L}$, il suffit de créer une machine M' qui simule M sur l'entrée souhaitée, puis d'en inverser la sortie : $M'(w)$ accepte si $M(w)$ rejette et $M'(w)$ rejette si $M(w)$ accepte. Cette construction peut être représentée comme suit :



- Si L_1 et L_2 sont décidables, alors $L_1 \cup L_2$ sont décidables.
  Preuve : Soit L_1 et L_2 deux langages décidables. Il existe donc deux machines M_1 et M_2 qui s'arrêtent toujours et telles que M_1 décide L_1 et M_2 décide L_2. On peut créer une machine M qui simule d'abord M_1 sur l'entrée souhaitée. Si $M_1(w)$ accepte, on peut déjà s'arrêter et accepter w (c'est suffisant). Si $M_1(w)$ refuse, on simule $M_2(w)$ et on renvoie la même réponse que $M_2(w)$.



- Si L_1 et L_2 sont décidables, alors $L_1 \cap L_2$ sont décidables.
  Preuve : Similaire à l'union, mais en acceptant seulement si les deux acceptent.



3.2.2 Langages reconnaissables

- Si L est reconnaissable, alors $\overline{L}$ peut ne pas être reconnaissable.
  Preuve : L est reconnaissable, donc il existe une machine M qui reconnaît L. Si $\overline{L}$ est reconnaissable, alors il existe aussi une machine M' qui reconnaît $\overline{L}$. On peut alors créer une machine M_D qui utilise M et M' pour décider L, ce qui est une contradiction pour certains langages reconnaissable qui ne sont pas décidables.
  Remarque : En fait, les langages décidables correspondent exactements aux cas particuliers des langages reconnaissables dont le complément est aussi reconnaissable.

- Si L_1 et L_2 sont reconnaissables, alors $L_1 \cup L_2$ est reconnaissable.
  Preuve (plus subtile) : Soit L_1 et L_2 deux langages reconnaissables. Il existe donc deux machines M_1 et M_2 tels que M_1 reconnaît L_1 et M_2 reconnaît L_2. On souhaiterait créer une machine M qui simule ces deux machines et qui accepte l'entrée w si au moins l'une des deux accepte. La difficulté est que si $w \notin L_1$, on ne peut pas attendre que M_1 termine pour tester si $w \in L_2$, car M_1 ne termine pas forcément dans ce cas. L'astuce consiste à simuler M_1 et M_2 en même temps. Par exemple, M peut simuler quelques pas de calcul sur M_1, puis quelques pas de calcul sur M_2, puis à nouveau M_1, et ainsi de suite en alternant les deux (oui, c'est possible !). Si $w \in L_1 \cup L_2$, l'une des deux machine finira par terminer en acceptant w, et M pourra à son tour terminer en acceptant w. Si $w \notin L_1 \cup L_2$, alors M ne terminera pas forcément, mais cela est OK puisqu'on souhaite seulement *reconnaître* $L_1 \cup L_2$.

- Si L_1 et L_2 sont reconnaissables, alors $L_1 \cap L_2$ est reconnaissable.
  Preuve : C'est plus facile que l'union. Il suffit de créer une machine M qui simule M_1 puis M_2 et qui n'accepte que si les deux acceptent. Si M_1 ou M_2 ne s'arrête pas, ce n'est pas grave, car cela veut dire que $w \notin L_1 \cap L_2$ et qu'il est donc valide de ne pas répondre (puisqu'on veut seulement *reconnaître* $L_1 \cap L_2$).

3.3 Réductions entre langages

Comment montrer qu'un langage (ou problème) est indécidable ? Nous avons déjà vu un exemple avec le langage de l'arrêt $L_H = \{(\langle M \rangle, w) \mid M \text{ termine sur l'entrée } w\}$ et la preuve que Turing a fait de son indécidabilité. La bonne nouvelle est qu'une fois que l'on dispose d'un langage indécidable, on peut s'en servir pour montrer que d'autres langages sont indécidables. Comment faire ? Supposons que l'on veuille montrer qu'un certain langage L est indécidable. Il suffit de montrer, par l'absurde, que si une machine existait pour décider L, alors cette machine pourrait nous permettre de décider L_H (ou tout autre langage indécidable). Comme L_H est indécidable, cela implique que L doit l'être aussi. Plus généralement, on parle de *réduction* entre langages.

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 permet de décider L_1. On écrit cela $L_1 \leq_T L_2$.

Lorsqu'on programme, on utilise ce concept en permanence. Par exemple si vous écrivez un programme qui résoud A en appelant une fonction qui résoud B, alors c'est que A se réduit à B. Et si par hasard A était indécidable, cela montrerait que B l'est aussi (puisque son utilisation permet de résoudre un problème indécidable).

3.3.1 Exemple

Soit le langage $L = \{(\langle M \rangle, w) \mid M \text{ accepte } w\}$. Décider ce langage revient à décider si une machine donnée répond OUI sur une entrée donnée. Autrement dit, on doit répondre OUI si $M(w)$ répond OUI, et NON si $M(w)$ boucle ou répond NON. Nous allons montrer que ce langage est indécidable en réduisant L_H à L, c.à.d. en montrant que s'il existe une machine qui décide L, on peut l'utiliser pour décider L_H.

Theorème 3.2. *L_H se réduit à L.*

Preuve. La preuve consiste à créer une machine M_H qui décide L_H en utilisant une hypothétique machine M_L capable de décider L. Nous allons aussi utiliser une machine universelle M_U capable de simuler $M(w)$, en l'adaptant pour qu'elle réponde OUI quand la simulation termine. Vous suivez ? On a bien trois machines. Il ne reste plus qu'à les utiliser !

Notre machine M_H reçoit en entrée les entrées normales du problème de l'arrêt, à savoir une machine $\langle M \rangle$ et une entrée w et doit décider si $M(w)$ termine. La machine M_U, qui doit simuler $M(w)$ ne sera jamais lancée. Nous allons seulement l'utiliser pour demander à M_L si M_U accepte $(\langle M \rangle, w)$, ce qui signifierait que $M(w)$ termine. Concrètement, nous allons appeler $M_L(\langle M_U \rangle, (\langle M \rangle, w))$ et renvoyer ce qu'elle renvoit (voir Figure 1).

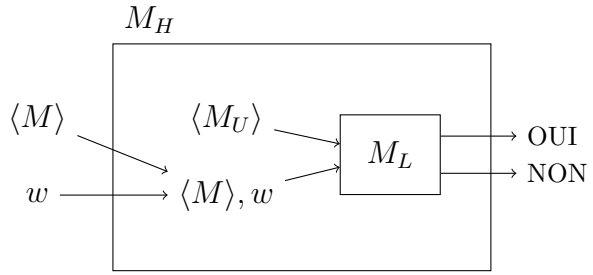


FIGURE 1 – Machine qui décide le langage de l'arrêt à l'aide d'une machine décidant L.

Pour terminer la preuve, vérifions que la réponse est correcte : M_H répond OUI si M_L répond OUI, ce qui implique que M_U accepte l'entrée $(\langle M \rangle, w)$. Par définition de M_U, cela implique bien que $M(w)$ termine. M_H répond NON si M_L répond NON, ce qui implique que M_U n'accepte pas $(\langle M \rangle, w)$. Cela ne peut arriver que si $M(w)$ ne termine pas. Les réponses de M_H reviennent donc bien à décider L_H, à supposer qu'une machine décidant L existe. □

Le fait que L_H est indécidable (Turing, 1936) et que $L_H \leq_T L$ (Théorème 3.2) nous permet maintenant de conclure que L est indécidable. La notion de réduction est centrale en informatique, y compris en algorithmique. Nous en reparlerons donc dans la partie complexité. Pour information, le langage L que nous avons montré indécidable 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 utilisée dans la preuve (dont l'usage ne correspondait pas à décider ou même reconnaître L_U).