TD 1.1 : Induction — suite et fin

1. Encore des définitions par induction

Décrire explicitement les ensembles définis par induction suivants :

— Pour $\Sigma = \{a, b, c\}$, soit $L_1 \subseteq \Sigma^*$ tel que $b \in L_1$ et si $u \in L_1$, alors $a \cdot u \cdot c \in L_1$.[1]
— Pour Σ un ensemble de symboles, soit $L_2 \subseteq \Sigma^*$ tel que $\varepsilon \in L_2$ et si $u \in L_2$ et $\ell, \ell' \in \Sigma$, alors $\ell \cdot u \cdot \ell' \in L_2$.
— Pour Σ un ensemble de symboles, soit $L_2 \subseteq \Sigma^*$ tel que $\Sigma \cup \{\varepsilon\} \subseteq L_3$ et si $u \in L_3$ et $\ell \in \Sigma$, alors $\ell \cdot u \cdot \ell \in L_3$.

— C'est l'ensemble $\{a^n b c^n \mid n \in \mathbb{N}\}$, c'est-à-dire l'ensemble des mots avec un certain nombre de a, puis un b, puis le même nombre de c que de a.
— C'est l'ensemble des mots de longueur paire.
— C'est l'ensemble des palindromes sur Σ.

2. La fonction d'Ackermann est bien définie

On reprend le principe d'induction bien fondée défini dans le TD 1.

2.1 Rappeler la définition de l'ordre lexicographique sur l'ensemble $\mathbb{N}^2$. *Dans la suite, on le note* $\leq_{lex}$.

$(n, m) \leq_{lex} (n', m')$ ssi $n < n'$, ou $n = n'$ et $m \leq m'$

2.2 Montrer par induction bien fondée que l'algorithme suivant *termine*, c'est-à-dire que la séquence d'instructions exécutées dans l'algorithme sur toute entrée est finie (pas de "boucles infinies") :

Pseudo-code

Données : $n, m \in \mathbb{N}$
Ackermann *(n, m)*
  si $n = 0$ **alors**
    retourner $m + 1$
  sinon si $m = 0$ **alors**
    retourner `Ackermann`$(n-1, 1)$
  sinon
    $k \leftarrow$ `Ackermann`$(m, n-1)$
    retourner `Ackermann`$(n-1,\ k)$

Indication : on peut utiliser dans cette question que $\leq_{lex}$ est bien fondé.

Montrons que l'hypothèse du principe d'induction est bien vérifiée par la propriété $\mathcal{P}_{n,m}$: "l'algorithme termine sur l'entrée (n, m)".
Supposons que pour tout couple $(n', m') <_{lex} (n, m)$, on a $\mathcal{P}_{n,m}$.

— Si $n = 0$, alors l'algorithme termine trivialement en ligne 4 en passant le premier test.
— Si $n \neq 0$ et $m = 0$, alors le premier test échoue mais le second est passé, ce qui nous fait arriver en ligne 6. Or, puisque $(n-1, 1) <_{lex} (n, 0) = (n, m)$, on a $\mathcal{P}_{n-1,1}$ qui est vraie par hypothèse d'induction : ainsi, l'appel récursif à l'algorithme sur l'entrée $(n-1, 1)$ termine, donc l'algorithme termine sur l'entrée (n, m) aussi.
— Sinon, l'algorithme ne passe pas les deux tests et arrive en ligne 8. On fait donc deux appels récursifs à l'algorithme. D'abord, le premier appel se fait sur l'entrée $(n, m-1) <_{lex} (n, m)$ donc il termine par hypothèse d'induction, et le second appel se fait sur l'entrée $(n-1, k) <_{lex} (n, m)$ qui termine aussi (même si k peut être très gros !).

Par le principe d'induction bien fondée, l'algorithme termine sur toute entrée (admissible).

2.3 Écrire explicitement la sortie de `Ackermann(`$1,k$`)`, `Ackermann(`$2,k$`)` et `Ackermann(`$3,k$`)` pour tout $k \in \mathbb{N}$.

2.4 Par récurrence sur $n \in \mathbb{N}$, montrer que `Ackermann(`n,m`)` $> m$ pour tout $m \in \mathbb{N}$.

La fonction d'Ackermann calculée par cet algorithme est l'exemple classique de fonction *non primitive récursive* : il n'existe pas d'algorithme (n'utilisant pas d'appels récursifs) calculant cette fonction sans utiliser de boucles non bornées, i.e. sans utiliser de **while**. Le montrer est cependant un exercice difficile.

1. $\cdot$ étant le symbole de concaténation de deux mots.