Corrections TD 02 – Machines de Turing et codage

Exercice 2. *L'arrêt*

Indiquer si chacun des énoncés qui suit est vrai ou faux, en justifiant.

1. $\nexists M_{halt}, \forall M, \forall w : M_{halt}(\langle M\rangle, w) = halt(\langle M\rangle, w)$.
   Vrai. Il s'agit de l'énoncé du théorème de l'arrêt : Il n'existe pas de machine de Turing M_{halt} qui calcule la fonction d'arrêt $halt$ des machines de Turing, c'est-à-dire telle que pour toute machine M et toute entrée w la machine M_{halt} lancée sur l'entrée $(\langle M\rangle, w)$ s'arrête avec la valeur $halt(\langle M\rangle, w)$ écrite sur le ruban.

2. $\forall M, \forall w, \nexists M_{halt} : M_{halt}(\langle M\rangle, w) = halt(\langle M\rangle, w)$.
   Faux. Soient les deux machines suivantes : $M_{idiote\,0}$ qui répond toujours 0 quelle que soit l'entrée, et $M_{idiote\,1}$ qui répond toujours 1 quelle que soit l'entrée. Alors $\forall M, \forall w$, on a $halt(\langle M\rangle, w) \in \{0, 1\}$ donc l'une de ces deux machines répond correctement, c'est-à-dire $halt(\langle M\rangle, w) = 0 = M_{idiote\,0}(\langle M\rangle, w)$ ou $halt(\langle M\rangle, w) = 1 = M_{idiote\,1}(\langle M\rangle, w)$. Ainsi $\forall M, \forall w$ il existe une machine M' (soit $M_{idiote\,0}$ soit $M_{idiote\,1}$) telle que $M'(\langle M\rangle, w) = halt(\langle M\rangle, w)$. Cette dernière affirmation est la négation de l'énoncé, qui est donc faux.

Exercice 4. *Codage*

On définit une fonction de codage $code_1$ des couples d'entiers de la manière suivante : étant donnés deux entiers x et y, on pose

$$code_1(x, y) = x_1 0 x_2 0 \dots x_{n-1} 0 x_n\, 1\, y_1 y_2 \dots y_m,$$

où $x_1 x_2 \dots x_n$ et $y_1 y_2 \dots y_m$ sont les représentations binaires respectives de x et y.

1. Ce codage est-il injectif ? (Autrement dit, un tel code correspond-il bien à un unique couple d'entiers ?)
   Oui. Dans le codage $x_1 0 x_2 0 \dots x_{n-1} 0 x_n\, 1\, y_1 y_2 \dots y_m$, la position du 1 « séparateur » est déterminée de manière unique : c'est la première occurrence de 1 en position paire. Un tel codage étant donné, on peut donc identifier ce séparateur, puis le préfixe $x_1 0 x_2 0 \dots x_{n-1} 0 x_n$ et le suffixe $y_1 y_2 \dots y_m$ qu'il sépare, et finalement les deux représentations binaires $x_1 x_2 \dots x_n$ et $y_1 y_2 \dots y_m$ de x et y.

2. Donnez le codage du couple $(10, 5)$.
   Les entiers 10 et 5 s'écrivent en binaire, respectivement, 1010 et 101. On intercale des 0 à l'intérieur de la représentation de 10 : 1000100, on ajoute un 1 séparateur entre les deux mots binaires ainsi obtenus et on conclut :
   $code_1(10, 5) = 10001001101$.

3. Quel est le couple associé au codage 100010100010111001110 ?
   Le premier 1 en position paire est en $14^{\text{ième}}$ position : 10001010001011001110. Il sépare donc les deux mots 1000101000101 et 1001110. On retire les 0 en positions paires dans le premier et on en déduit que x et y sont les entiers de représentations binaires 1011011 et 1001110, c'est-à-dire les entiers 91 et 78.

4. Quel est le nombre de bits utilisés pour coder un couple (x, y) ?
   Avec les 0 intercalés dans x et le 1 séparateur, on ajoute autant de symboles qu'il y en a

dans l'écriture binaire de x. Si l'on convient de noter $|u|$ la longueur d'un mot et $\text{bin}(n)$ le codage binaire d'un entier n, on obtient donc : $|\mathit{code}_1(x,y)| = 2|\text{bin}(x)| + |\text{bin}(y)|$. Or on sait que l'écriture binaire d'un entier n est de longueur $|\text{bin}(n)| = \lfloor \log n \rfloor + 1$. Il s'ensuit que $|\mathit{code}_1(x,y)| = 2\lfloor \log x \rfloor + \lfloor \log y \rfloor + 3$.

On définit un nouveau codage code_2 en posant maintenant

$$\mathit{code}_2(x,y) = t_1 0 t_2 0 \dots t_{\ell-1} 0 t_\ell \, 1 \, x_1 x_2 \dots x_n \, y_1 y_2 \dots y_m,$$

où $x_1 x_2 \dots x_n$ et $y_1 y_2 \dots y_m$ sont les représentations binaires respectives de x et y, et où $t_1 t_2 \dots t_\ell$ est la représentation binaire de l'entier n (qui est lui-même la taille de la représentation binaire de x).

5. Ce codage est-il injectif ?
Oui. Comme précédemment, le 1 séparateur est identifiable comme le premier 1 en position paire. On peut alors extraire du code le préfixe $t_1 0 t_2 0 \dots 0 t_\ell$ et le suffixe $x_1 x_2 \dots x_n \, y_1 y_2 \dots y_m$. Du préfixe, on déduit l'entier n de représentation binaire $t_1 t_2 \dots t_\ell$. On peut alors extraire les n premiers caractères du mot $x_1 x_2 \dots x_n \, y_1 y_2 \dots y_m$, qui fournissent l'écriture binaire de x. Les symboles restant forment l'écriture binaire de y.

6. Donnez le codage du couple $(10, 5)$.
On calcule d'abord $\text{bin}(10) = 1010$ et $\text{bin}(5) = 101$. La longueur de $\text{bin}(10)$ est donc 4, qui s'écrit 100 en binaire. On insère des 0 à l'intérieur de $\text{bin}(4) : 1000$; on fait suivre le mot obtenu d'un 1 puis des écritures binaires de 10 et 5 prises dans cet ordre. On obtient finalement : $\mathit{code}_2(10,5) = 100001101010101$.

7. Quel est le couple (x, y) associé au codage 100011100011001 ?
On identifie le premier 1 en position paire : 100011100011001. On récupère le préfixe et le suffixe séparés par ce 1 : 10001 et 100011001. Dans le préfixe, on retire les 0 en positions paires. On obtient le mot 101, qui est l'écriture binaire de l'entier 5. On en déduit que dans le suffixe 100011001, les 5 premiers caractères forment la représentation binaire de x, et les suivants, la représentation binaire de y. Autrement dit, $\text{bin}(x) = 10001$ et $\text{bin}(y) = 1001$. Finalement, $(x, y) = (17, 9)$.

8. Quel est le nombre de bits utilisés pour coder un couple (x, y) ?
Clairement, $|\mathit{code}_2(x,y)| = 2|\text{bin}(|\text{bin}(x)|)| + |\text{bin}(x)| + |\text{bin}(y)|$, d'où, en approximant la longueur de $\text{bin}(n)$ par $\log n + 1$: $|\mathit{code}_2(x,y)| \approx 2\log(\log x + 1) + \log x + \log y + 4$.

On souhaite généraliser ce codage aux tuples d'entiers $(x^1, \dots, x^k)$ pour k quelconque.

9. Proposez un tel codage.
Un choix possible est de concaténer les représentations binaires des k entiers du tuple $(x^1, \dots, x^k)$ à coder. On obtient le mot $\text{bin}(x^1)\text{bin}(x^2)\dots\text{bin}(x^k)$. Pour identifier la portion de ce mot correspondant à chaque entier, on le préfixe par le nombre k de mots à extraire. Mais cela ne suffit pas : il y a bien des manières de découper $\text{bin}(x^1)\text{bin}(x^2)\dots\text{bin}(x^k)$ en k facteurs. On ajoute donc comme information, comme on l'a fait avec code_2, la longueur en binaire de chaque entier à identifier : $\ell_1 = |\text{bin}(x^1)|$, …, $\ell_k = |\text{bin}(x^k)|$. L'entier k, comme chaque ℓ_i, est écrit en binaire en insérant des 0 à chaque place paire et un 1 à fin du code de chacun de ces entiers, pour permettre sa lecture, comme dans les codes précédents.

Finalement, un tuple $(x^1, \dots, x^k)$ est codé par :

$$\mathit{code}_3(x^1, \dots, x^k) = \text{bin}(k)^+ \, \text{bin}(\ell_1)^+ \dots \text{bin}(\ell_k)^+ \, \text{bin}(x^1) \dots \text{bin}(x^k)$$

où ℓ_i désigne la longueur de $\text{bin}(x^i)$ et où l'on a noté $\text{bin}(n)^+$ le codage binaire d'un entier n enrichi par des 0 intercalés et un 1 final, comme dans les codes précédents. (Par exemple, $\text{bin}(5) = 101$ et $\text{bin}(5)^+ = 10001$.)

Notons que $code_3$ est bien un code injectif :
— le premier 1 en position paire définit un préfixe correspondant à l'entier k ;
— l'identification de cet entier permet d'extraire les k longueurs $\ell_1, \dots, \ell_k$, encadrées par les k prochains 1 en positions paires ;
— la détermination de ces k longueurs permet alors de découper la fin du mot en k facteurs correspondant aux k entiers $x^1, \dots, x^k$.

10. Donnez le codage du tuple $(6, 11, 10, 3)$.
On calcule d'abord :

(a) Les représentations binaires des entiers $6, 11, 10, 3$ du tuple : 110, 1011, 1010, 11.

(b) Les longueurs de ces représentations : 3, 4, 4, 2.

(c) Les codages de ces longueurs :
— $\text{bin}(3) = 11$ d'où $\text{bin}(3)^+ = 1011$;
— $\text{bin}(4) = 100$ d'où $\text{bin}(4)^+ = 100001$;
— $\text{bin}(2) = 10$ d'où $\text{bin}(2)^+ = 1001$.

(d) Le nombre d'entiers dans le tuple : 4, et son codage : $\text{bin}(4)^+ = 100001$.

Le code de $(6, 11, 10, 3)$ est alors obtenu par concaténation des codes du nombre d'entiers, de leurs longueurs, des entiers eux-mêmes. Cela donne :

$$code_3(6, 11, 10, 3) = 1000011000011000011000011011110101110101101.$$

11. Donnez le nombre de bits utilisés pour coder un tuple arbitraire $(x^1, \dots, x^k)$.
Convenons de noter $\lg(x)$ la longueur du codage binaire d'un entier x. On calcule facilement :

$$|code_3(x^1, \dots, x^k)| = \lg(k) + \lg(\lg(x^1)) + \cdots + \lg(\lg(x^k)) + \lg(x^1)) + \cdots + \lg(x^k)).$$

Exercice 6. *MT : multi-ruban*

Objectif : montrer que le modèle des machines de Turing à plusieurs rubans et plusieurs têtes de lecture (une tête de lecture indépendante par ruban, et un seul état pour toute la machine) est équivalent au modèle des machines de Turing. Le multi-ruban est très pratique !

En fonction de l'état et du symbole lu sur chacun des rubans, la machine peut
— changer d'état,
— écrire un symbole sur chaque ruban,
— déplacer chaque tête vers la droite ou la gauche indépendamment les unes des autres.

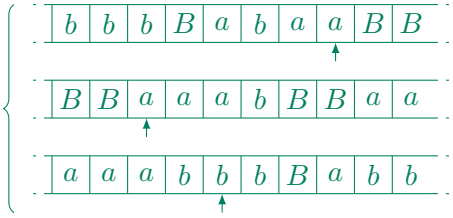
1. Donner le type de la fonction de transition δ, et donner un exemple de transition.
$\delta : (Q \setminus \{q_F\}) \times \Gamma^k \to Q \times \Gamma^k \times \{R, L\}^k$ avec k le nombre de rubans.

Dans l'état initial, l'entrée est écrite sur le premier ruban et tous les autres rubans sont vides. Un mot est accepté si et seulement si la machine entre dans l'état final q_F au cours du calcul.

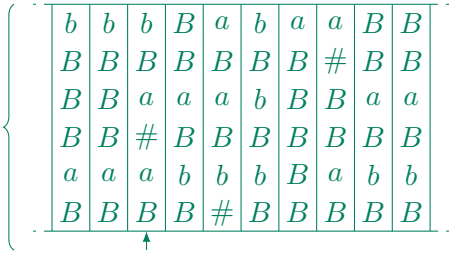
2. Comment simuler une MT *multi-ruban* avec une MT *mono-ruban* ?
L'idée est de simuler une machine à k rubans et k têtes sur un alphabet Γ dont le symbole blanc est B, avec une machine mono-ruban sur l'alphabet $\Gamma' = (\Gamma \times \{B, \#\})^k \cup \Sigma$ dont le symbole blanc est (B, B, B, B, B, B). L'ajout de Σ sert à ce que la machine mono-ruban puisse être lancée sur les mêmes entrées que la machine multi-ruban, afin de reconnaître

le même langage (voir la fin de la construction). On peut voir *un* symbole de la machine mono-ruban comme $2k$ sous-symboles (k couples de $\Gamma \times \{B, \#\}$). Chacun des k rubans avec sa tête sera représenté par deux sous-symboles : le premier correspond au contenu de ce ruban et sur le second nous avons un $\#$ à l'endroit où se trouve la tête de lecture de ce ruban (B partout ailleurs). Voici un exemple pour $k = 3$ rubans :

machine à 3 rubans dans l'état q



machine mono-ruban dans l'état q

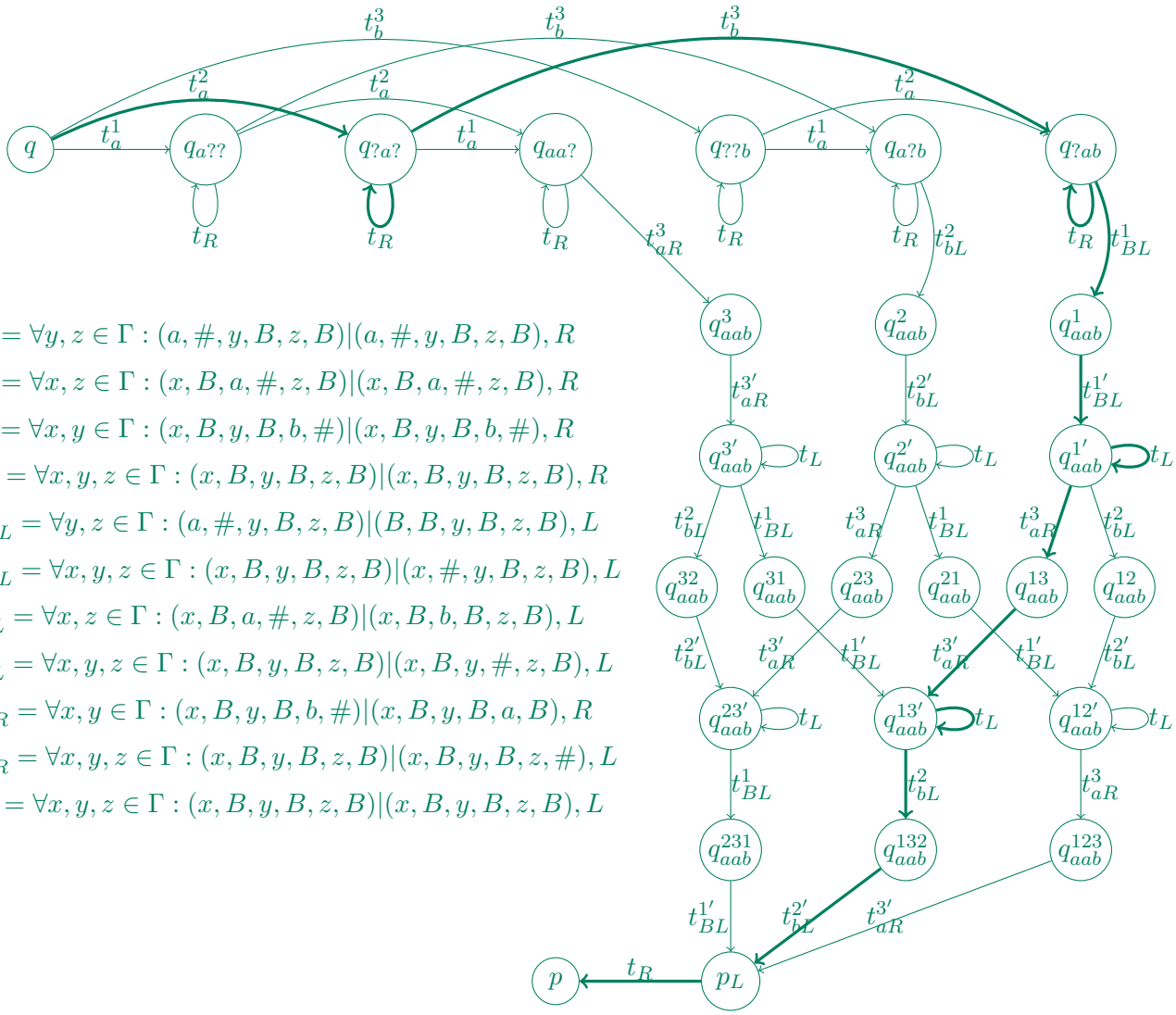


Remarquons par exemple que la case sous la tête de lecture de la machine mono-ruban de l'exemple contient le symbole $(b, B, a, \#, a, B)$ de Γ'. Une astuce importante consiste à toujours ramener l'unique tête de lecture/écriture de la machine mono-ruabn (qui simule la machine à k rubans) sur la case contenant la tête la plus à gauche de la machine multi-ruban simulée. Alors pour simuler *une* étape de la machine à k-rubans, la machine mono-ruban fait un aller-retour jusqu'à la tête la plus à droite :

— sur l'aller la machine mono-ruban se souvient dans son état des sous-symboles sous les têtes de chaque ruban simulé, une fois arrivée à la fin de l'aller ($k^{\text{ième}}$ sous-symbole $\#$ trouvé) la machine mono-ruban sait donc quelle transition de la machine à k rubans elle doit simuler,

— sur le retour la machine mono-ruban met à jour tous les sous-synmboles de ruban et tous les sous-symboles de têtes selon la transition à simuler, dans l'ordre où elle les rencontre, en s'arrêtant sur la tête la plus à gauche ($k^{\text{ième}}$ sous-symbole $\#$ trouvé).

Par exemple, pour simuler la transition $\delta(q, a, a, b) = (p, B, b, a, L, L, R)$ de la machine à $k = 3$ rubans, la machine mono-ruban aura les transitions suivantes :



$t_a^1 = \forall y, z \in \Gamma : (a, \#, y, B, z, B) | (a, \#, y, B, z, B), R$

$t_a^2 = \forall x, z \in \Gamma : (x, B, a, \#, z, B) | (x, B, a, \#, z, B), R$

$t_b^3 = \forall x, y \in \Gamma : (x, B, y, B, b, \#) | (x, B, y, B, b, \#), R$

$t_R = \forall x, y, z \in \Gamma : (x, B, y, B, z, B) | (x, B, y, B, z, B), R$

$t_{BL}^1 = \forall y, z \in \Gamma : (a, \#, y, B, z, B) | (B, B, y, B, z, B), L$

$t_{BL}^{1'} = \forall x, y, z \in \Gamma : (x, B, y, B, z, B) | (x, \#, y, B, z, B), L$

$t_{bL}^2 = \forall x, z \in \Gamma : (x, B, a, \#, z, B) | (x, B, b, B, z, B), L$

$t_{bL}^{2'} = \forall x, y, z \in \Gamma : (x, B, y, B, z, B) | (x, B, y, \#, z, B), L$

$t_{aR}^3 = \forall x, y \in \Gamma : (x, B, y, B, b, \#) | (x, B, y, B, a, B), R$

$t_{aR}^{3'} = \forall x, y, z \in \Gamma : (x, B, y, B, z, B) | (x, B, y, B, z, \#), L$

$t_L = \forall x, y, z \in \Gamma : (x, B, y, B, z, B) | (x, B, y, B, z, B), L$

La partie supérieurs (état $q_{a??}, q_{?a?}, \ldots, q_{?ab}$) concerne l'aller (vers la droite), et lorsqu'on a connaissance des 3 sous-symboles sous les têtes de la machine multi-ruban, on transitionne vers la partie inférieure qui s'occupe au retour (vers la gauche) d'écrire les nouveaux sous-symboles et de déplacer les têtes comme l'aurait fait la machine multi-ruban. Vous pouvez vérifier que cela mène la machine mono-ruban à :

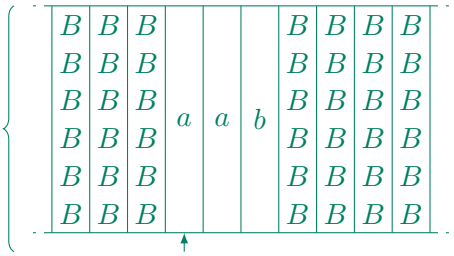
machine mono-ruban dans l'état p

b	b	b	B	a	b	a	B	B	B
B	B	B	B	B	B	$\#$	B	B	B
B	B	b	a	a	b	B	B	a	a
B	$\#$	B	B	B	B	B	B	B	B
a	a	a	b	a	b	B	a	b	b
B	B	B	B	B	$\#$	B	B	B	B

Attention : la machine mono-ruban doit gérer toutes les combinaisons possibles de positions des têtes ! Notamment les cas où plusieurs sous-symboles # apparaissent dans un même symbole de Γ ou sur des cases adjacentes, qui ne sont pas gérés sur l'exemple.
Pour construire entièrement la machine mono-ruban qui simule une machine multi-ruban donnée, il faut assembler ce mécanisme pour toutes les transitions de la machine multi-ruban. Il faut également ajouter quelques transitions pour initialiser la simulation, afin de

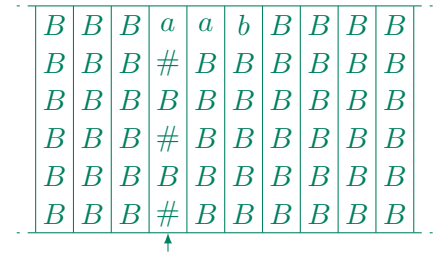
passer de la configuration de départ, par exemple sur l'entrée aab :

départ de la machine mono-ruban



en une configuraiton où la simulation est initialisée, comme cela :

machine mono-ruban initialisée



Remarque : la simulation est un peu lente (quadratique, car la longueur des aller-retours effectués par la machine mono-ruban est bornée par le temps de calcul de la machine multi-ruban), et la machine mono-ruban a énormément d'états et un très gros alphabet de ruban, mais du point de vue de la calculabilité (tous les ensembles sont finis et le langage reconnu est identique), c'est une simulation d'une machine multi-ruban par une machine mono-ruban tout à fait valide !