Sur la complexité de programmes universels

Alain Colmerauer*

18 octobre 2004

Résumé

Ce papier est consacré à la définition et au calcul de la complexité de différents programmes universels U pour différentes machines. Cette complexité est d'abord vue comme le nombre moyen d'instructions exécutées par U pour simuler l'exécution d'une instruction d'un programme P ayant pour donnée x.

Pour définir une complexité ne dépendant ni de P ni de x on introduit alors la notion de coefficient d'introspection qui exprime le nombre moyen d'instructions exécutées par U pour simuler l'exécution d'une de ses propres instructions. Nous montrons comment obtenir ce coefficient en calculant une matrice carré dont les éléments sont des nombres d'instructions exécutées lors du déroulement de parties sélectionées du U sur des données sélectionnées. Ce coefficient est alors égal à la plus grande valeur propre de cette matrice.

Nous illustrons l'approche par deux exemples de programmes universels particulièrement efficaces : l'un pour une machine de Turing à trois symboles (symbole blanc non compris) avec un coefficient d'introspection de 3682,98, l'autre pour une machine arthmétique à adressage indirect avec un coefficient d'introspection de 26,27.

* Laboratoire d'Informatique Fondamentale de Marseille, CNRS et Universités de Provence et de la Méditerranée

Table des matières

1 Introduction

Pour familiariser les étudiants de deuxième année d'Université avec la programmation de bas niveau, j'ai commencé il y a quelques années à leur enseigner la programmation de machines de Turing. L'exercice principal consistait à terminer et à tester un programme universelle dont je décrivais l'architecture et donnais plusieurs parties. Les tests se révélèrent décevants, le programme universel étant trop lent pour simuler l'exécution de programmes conséquents. Entre autres il était impossible de le faire tourner sur son propre code, au sens où nous l'entendons à la section 4. Au fil des années j'ai donc était amené à concevoir un prgramme universel de plus plus rapide, quoique de plus en plus compliquée, pouvant tourner sur lui-même en un temps raisonnable. Pour simuler l'exécution d'une de ses propres instructions il exécute en moyenne 3683,98 instructions, ou plus exactement son *coefficient d'introspection*, notion centrale de ce papier, est de 3683,98.

Ce papier présente ce résultat en le plaçant dans un cadre général concernant d'autre machines que les machines de Turing. Il est organisé en 5 sections suivies d'un appendice. La première section constitue cette introduction et la dernière la conclusion. La section 2 précises les notions de machine programmée, machine, programme, transition, instruction et la section 3 les illustres sur trois exemples : les machines de Turing, les machines de Turing à direction interne, une variante des machines précedentes, une machine arithmétique à adressage indirect. A la section 4, la partie principal, on montre comment vérifier l'existence d'un coefficient d'introspection et le calculer. La démonstration du théorème qui y est présenté nécessiterait plusieur pages. Comme la pluspart des démonstration elle est omise par manque de place. Une version plus complète du papier, avec toutes les démonstrations, est en préparation. Les sections 5 et 6 sont consacrées à deux programmes universels particulièrement efficaces, l'un, déjà mentionné, pour une machine de Turing, l'autre pour une machine arithmétique à adressage indirect.

Nous ne connaissons pas d'autres travaux sur la conception de programmes universels efficaces. Citons cependant les travaux bien connus de M. Minsky [1] et Y. Rogozin [3] sur la conception de programmes universels, pour machine de Turing, comportant un très petit nombre d'états. Curieusement ils se révèlent particulièrement inefficaces en terme de nombres d'instructions exécutées.

2 Machines

2.1 Définitions de base

Définition 1 Une *machine programmée* est un couple $(\mathcal{M},P)$, où $\mathcal{M}$ est une *machine* et P un *programme* pour $\mathcal{M}$.

Définition 2 Une *machine* $\mathcal{M}$ est un quintuplet $(\Sigma,C,\alpha,\omega,\mathcal{I})$, où

- Σ, *l'alphabet* de $\mathcal{M}$, est un ensemble fini non vide ;
- C, est un ensemble, généralement infini, de *configurations* ; les couples (c,c') d'éléments de C sont appelés *transitions* ;
- α, *la fonction d'entrée* associe une configuration $\alpha(x)$ à chaque élément x de $\Sigma^\star$;
- ω, *la fonction de sortie* associe un élément $\omega(c)$ de $\Sigma^\star$ à chaque configuration c ;
- $\mathcal{I}$, est un ensemble dénombrable d'instructions, une *intruction* étant un ensemble de transitions *compatibles*, c'est-à-dire, ayant des première composantes toutes distinctes.

Définition 3 Un *programme* P pour une telle machine $\mathcal{M}$ est un sous-ensemble fini de l'ensemble $\mathcal{I}$ de ses instructions, tel que les transitions de $\bigcup P$ soient compatibles.[1] L'ensemble des configurations intervenant dans P est notée C_P. Une configuration c de $\mathcal{M}$ est dite *finale pour* P, s'il n'existe aucune transition de $\bigcup P$ qui commence par c.

2.2 Fonctionnement

Donnons nous un machine $\mathcal{M} = (\Sigma,C,\alpha,\omega,\mathcal{I})$ et un programme P pour $\mathcal{M}$. Le fonctionnement de la machine programmée $(\mathcal{M},P)$ s'explique par le schéma

$$\begin{array}{ccccccccccc} x & & & & & & & & & & y \\ \downarrow & & & & & & & & & & \uparrow \\ c_0 & \longrightarrow & c_1 & \longrightarrow & c_2 & & \cdots & & c_{n-1} & \longrightarrow & c_n \end{array}$$

et plus précisément par la définition des fonctions suivantes[2], où x est un mot sur Σ et c,c' des configurations de C :

$$orbite_{\mathcal{M}}(P,x) = \begin{cases} \text{la plus longue suite possible } c_0,c_1,c_2,\ldots \text{ avec} \\ c_0 = \alpha(x) \text{ et chaque } (c_i,c_{i+1}) \text{ élément de } \bigcup P, \end{cases}$$

$$out_{\mathcal{M}}(P,x) \quad = \begin{cases} \nearrow, & \text{si } orbite(P,x) \text{ est infini,} \\ \omega(c_n), & \text{si } orbite(P,x) \text{ se termine par } c_n \end{cases}$$

et, pour pouvoir parler de complexité,

$$trace_{\mathcal{M}}(P,c,c') = \begin{cases} \text{la suite la plus courte de la forme} \\ (c_0,c_1)\,(c_1,c_2)\,(c_2,c_3)\,\ldots\,(c_{n-1},c_n), \text{ avec} \\ c = c_0, \text{ chaque } (c_i,c_{i+1}) \in \bigcup P,\ c_n = c', \text{ si elle existe,} \\ \\ \nearrow, \text{ si elle n'existe pas,} \end{cases}$$

$$trace_{\mathcal{M}}(P,x) \quad = \begin{cases} \text{la plus longue suite de la forme} \\ (c_0,c_1)\,(c_1,c_2)\,(c_2,c_3)\,\ldots \\ \text{avec } orbite(P,x) = c_0,c_1,c_2,\ldots \end{cases}$$

$$cout_{\mathcal{M}}(P,x) \quad = \begin{cases} \infty, & \text{si } trace(P,x) \text{ est infinie,} \\ |trace(P,x)|, & \text{si } trace(P,x) \text{ est finie,} \end{cases}$$

où $|trace(P,x)|$ désigne la longueur de la suite finie $trace(P,x)$.

1. P étant un ensemble d'ensembles $\bigcup P$ désigne l'ensemble des éléments qui appartiennent à au moins un élément de P et donc ici l'ensemble des transitions intervenant dans le programme P.

2. Quand il n'y a pas d'ambigüité, nous omettons l'indice $\mathcal{M}$.

2.3 Simulation d'une machine par une autre

Soient $\mathcal{M}_1$ et $\mathcal{M}_2$ deux machines de même alphabet Σ et soient $\mathcal{P}_1$ et $\mathcal{P}_2$ l'ensemble de leurs programmes.

Définition 4 Deux machines programmées de la forme $(\mathcal{M}_1,P_1)$ et $(\mathcal{M}_2,P_2)$ sont *équivalentes* si, pour tout $x \in \Sigma^\star$, à la fois :

$$\begin{array}{rcl} out_{\mathcal{M}_1}(P_1,x) & = & out_{\mathcal{M}_2}(P_2,x), \\ cout_{\mathcal{M}_1}(P,x) & = & cout_{\mathcal{M}_2}(P_2,x). \end{array}$$

Définition 5 La transformation de programmes $f : \mathcal{P}_1 \to \mathcal{P}_2$ *simule* $\mathcal{M}_1$ sur $\mathcal{M}_2$ si, pour tous $P_1 \in \mathcal{P}_1$, les machines programmées $(\mathcal{M}_1,P_1)$ et $(\mathcal{M}_2,f(P_1))$ sont équivalentes.

3 Exemples de machines

3.1 Machines de Turing

Informellement il s'agit de machines de Turing classiques avec un ruban bi-infini et des instructions notées $[q_i,abd,q_j]$, avec $d = G$ ou $d = D$, et signifiant : si la machine est dans l'état q_i et que le symbole lu par la tête de lecture-écriture est a, elle remplace a par b, puis déplace sa tête d'un symbol à gauche ou à droite, suivant que $d = G$ ou $d = D$, et passe dans l'état q_j. Au départ tout le ruban est rempli de blancs sauf une portion finie qui contient la donnée intiale, la tête d'écriture-lecture étant positionnée sur la case qui précède cette donnée. Lorsqu'il n'y a plus d'instructions à exécuter la machine produit pour résultat le plus long mot qui ne contient pas de blancs et qui est situé à droite de la tête de lecture-écriture.

Formellement on se donne tout d'abord un ensemble infini dénombrable $\{q_1,q_2,\ldots\}$ *d'états* et un symbole spécial u, le *blanc*. Pour tout mot sur un alphabet de la forme $\Sigma \cup \{\mathsf{u}\}$ on note $\cdot x$ le mot x amputé des blancs du début et $x\cdot$ le mot x amputé des blancs de la fin.

Définition 6 Une machine $(\Sigma,C,\alpha,\omega,\mathcal{I})$ est dite *de Turing* si,

- $\Sigma \neq \Sigma_\mathsf{u}$, avec $\Sigma_\mathsf{u} = \Sigma \cup \{\mathsf{u}\}$,
- C est l'ensemble des quadruplets de la forme $[q_i, \cdot x,a,y\cdot]$, avec q_i un état quelconque, x,y pris dans $\Sigma^\star_\mathsf{u}$ et a pris dans Σ_u,
- $\alpha(x) = [q_1,\varepsilon,\mathsf{u},x]$, pour tout $x \in \Sigma^\star$,
- $\omega([q_i, \cdot x,a,y\cdot])$ est l'élément de $\Sigma^\star$ le plus long qui commence $y\cdot$,
- $\mathcal{I}$ est l'ensembles des instructions notées et définies, pour tous les états q_i,q_j et éléments a,b de Σ_u, par
  $[q_i, abG, q_j] \stackrel{\text{def}}{=} \{([q_i, \cdot xc,a,y\cdot], [q_j, \cdot x,c,by\cdot]) \mid (x,c,y) \in E\}$,
  $[q_i, abD, q_j] \stackrel{\text{def}}{=} \{([q_i, \cdot x,a,cy\cdot], [q_j, \cdot xb,c,y\cdot]) \mid (x,c,y) \in E\}$,
  avec $E = \Sigma^\star_\mathsf{u} \times \Sigma_\mathsf{u} \times \Sigma^\star_\mathsf{u}$,

3.2 Machines de Turing avec direction interne

Il s'agit d'une variante des machines de Turing précédemment décrite ayant une direction interne de déplacement de la tête de lecture-écriture égale à gauche-droite au départ. Les instructions sont notées $[q_i, abs, q_j]$, avec $s = +$ ou $s = -$ et signifient : si la machine est dans l'état q_i et que

le symbole lu par la tête de lecture-écriture est a, la machine remplace a par b, conserve ou change la la direction par défaut suivant que $s = +$ ou $s = -$, déplace sa tête d'une case dans la nouvelle direction par défaut, et passe dans l'état q_j.

Formellement en reprenant les ingrédients de la machine de Turing classique, on aboutit à :

Définition 7 Une machine $(\Sigma,C,\alpha,\omega,\mathcal{I})$ est dite *de Turing avec direction interne* si,

- $\Sigma \neq \Sigma_{\mathsf{u}}$, avec $\Sigma_{\mathsf{u}} = \Sigma \cup \{\mathsf{u}\}$,
- C est l'ensemble des quintuplets de la forme $[d,q_i,\cdot x,a,y\cdot]$, avec $d \in \{G,D\}$, q_i un état quelconque, x,y pris dans $\Sigma^\star_{\mathsf{u}}$ et a pris dans Σ_{u},
- $\alpha(x) = [D,q_1,\varepsilon,\mathsf{u},x]$, pour tout $x \in \Sigma^\star$,
- $\omega([d,q_i,\cdot x,a,y\cdot])$ est l'élément de $\Sigma^\star$ le plus long qui commence $y\cdot$,
- $\mathcal{I}$ est l'ensembles des instructions notées et définies, pour tous les états q_i,q_j, tous les éléments a,b de Σ_{u} et tout $s \in \{+,-\}$, par
  $[q_i, abs, q_j] \stackrel{\text{def}}{=}$
  $\{([d,q_i,\cdot xc,a,y\cdot],[G,q_j,\cdot x,c,by\cdot]) \mid (d,s) \in E_1 \text{ et } (x,c,y) \in F\} \cup$
  $\{([d,q_i,\cdot x,a,cy\cdot],[D,q_j,\cdot xb,c,y\cdot]) \mid (d,s) \in E_2 \text{ et } (x,c,y) \in F\}$,
  avec $E_1 = \{(G,+),(D,-)\}$, $E_2 = \{(D,+),(G,-)\}$ et $F = \Sigma^\star_{\mathsf{u}} \times \Sigma_{\mathsf{u}} \times \Sigma^\star_{\mathsf{u}}$,

3.3 Liens entre les deux types de machines de Turing

Soient $\mathcal{M}_1$ et $\mathcal{M}_2$ des machines de même alphabet Σ, l'une de Turing, l'autre de Turing avec direction interne et soient $\mathcal{P}_1$ et $\mathcal{P}_2$ l'ensembles de leurs programmes. Comme nous allons le voir il existe des liens étroits entre ces deux types de machines et nous en tirerons parti par la suite. Tout d'abords on montre que :

Propriété 1 *Les transformations de programmes $g_1 : \mathcal{P}_1 \to \mathcal{P}_2$ et $g_2 : \mathcal{P}_2 \to \mathcal{P}_1$, définies ci-dessous, simulent $\mathcal{M}_1$ sur $\mathcal{M}_2$ et $\mathcal{M}_2$ sur $\mathcal{M}_1$:*

$$g_1(P_1) \stackrel{\textit{def}}{=} \{[q_{2i-h}, abd, q_{2j-k}] \mid [q_i, abs, q_j] \in P' \textit{ et } (h,k,d,s) \in E\},$$
$$g_2(P_2) \stackrel{\textit{def}}{=} \{[q_{2i-h}, abs, q_{2j-k}] \mid [q_i, abd, q_j] \in P \textit{ et } (h,k,s,d) \in E\},$$

avec

$$E = \begin{Bmatrix}(1,1,+,D)\\(1,0,-,G)\end{Bmatrix} \cup \begin{Bmatrix}(0,1,-,D)\\(0,0,+,G)\end{Bmatrix}.$$

Introduisons maintenant les notions suivantes concernant un programme P_ℓ quelconque pour $\mathcal{M}_\ell$, avec $\ell = 1$ ou $\ell = 2$.

Définition 8 Un état q_i est dit *accessible dans P_ℓ*, s'il est égal à q_1 ou s'il existe un sous-ensemble de P_ℓ de la forme :

$$\{[q_{k_0}, a_1b_1e_1, q_{k_1}],[q_{k_1}, a_2b_2e_2, q_{k_2}], \ldots,[q_{k_{m-1}}, a_mb_me_m, q_{k_m}]\},$$

avec $k_0 = 1$ et $k_m = i$, les a_i et b_i étant prix dans Σ et les e_i étant pris dans $\{D,G\}$ ou $\{+,-\}$, suivant que $\ell = 1$ ou $\ell = 2$.

Définition 9 Pour tout sous-ensemble de P_ℓ de la forme

$$\{[q_{k_0}, a_1b_1e_1, q_{k_1}],[q_{k_1}, a_2b_2e_2, q_{k_2}], \ldots ,[q_{k_{m-1}}, a_mb_me_m, q_{k_m}]\},$$

avec les a_i et b_i prix dans Σ et, suivant le cas, les e_i pris dans $\{D,G\}$ ou $\{+,-\}$, la suite de triplets $a_1b_1e_1$,$a_2b_2e_2, \ldots ,a_mb_me_m$ est dite *effet potentiel* de l'état q_{k_0}. Deux états de P_ℓ qui ont le même ensemble d'effets potentiels sont dit *fusionnables*.

Définitions 10

- $accessible(P_\ell)$ *désigne l'ensemble des instructions de* P_ℓ *faisant intervenir des états accessibles dans* P_ℓ,
- $fusion(P_\ell)$ *désigne le programme obtenu en remplaçant dans* P_ℓ, *et en parallèle, tout occurrence d'état* q_i *par l'état* q_j *de plus petit indice possible qui est tel que* q_i *et* q_j *soient fusionnables,*
- $compact(P_\ell)$ *désigne le programme obtenu en renommant chaque état* q_i *de* P_ℓ *en* $q_{i'}$ *de façon à ce que les entiers* i' *soient les plus petits possibles et que* $i < j$ *entraîne* $i' < j'$,
- $propre(P_\ell) := compact(accessible(fusion(P_\ell)))$.

On montre que pour $\ell = 1$ et $\ell = 2$:

Propriété 2 *Les machines programmées* $(\mathcal{M}_\ell,propre(P_\ell))$ *et* $(\mathcal{M}_\ell,P_\ell)$ *sont equivalentes.*

On montre aussi que :

Théorème 1 *Les transformations de programmes* $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ *et* $f_2 : \mathcal{P}_2 \to \mathcal{P}_1$, *définies par* $f_\ell = propre \circ g_\ell$, *avec* g_ℓ *défini dans la propriété 1, simulent* $\mathcal{M}_1$ *sur* $\mathcal{M}_2$ *et* $\mathcal{M}_2$ *sur* $\mathcal{M}_1$ *et sont telles que* $f_2 \circ f_1 \circ f_2 \circ f_1 = f_2 \circ f_1$ *et* $f_1 \circ f_2 \circ f_1 \circ f_2 = f_1 \circ f_2$.

3.4 Machine arithmétique à adressage indirecte

Il s'agit d'une machine ayant une infinité de régistres r_0,r_1,$r_2, \ldots$. Chaque registre contient un entier positif ou nul, aussi grand que l'on veut. Chaque instruction commence par un numéro et la machine éxécute chaque fois l'instruction dont le numéro est contenu dans r_0 et, sauf cas exceptionnel, augmente r_0 de 1. Il y a cinq types d'instruction : introduction d'une constante dans un régistre, addition et soustraction d'un régistre à un autre, deux types de transferts indirects d'un registre à un autre et test à zéro du contenu d'un régistre.

D'une façon plus précise,

Définition 11 Cette machine $(\Sigma,C,\alpha,\omega,\mathcal{I})$ est définie par,

- $\Sigma = \{c_1, \ldots ,c_m\}$,
- C est l'ensemble des suite infinies d'entiers naturels de la forme $r = (r_0,r_1,r_2, \ldots)$,
- $\alpha(a_1 \ldots a_n) = (0,25,1, \ldots ,1,r_{24+1}, \ldots ,r_{24+n},0,0, \ldots)$, with r_{24+i} égal à $1, \ldots ,m$ suivant que a_i est égal à $c_1, \ldots ,c_m$,
- $\omega(r_0,r_1, \ldots) = a_1 \ldots a_n$, avec a_i égal à $c_1, \ldots ,c_m$ suivant que $\overline{r_{r_1+i}}$ égale $1, \ldots ,m$, et sous réserve que n soit le plus grand entier tel que $r_{r_1}, \ldots ,r_{r_1+n}$ soient des éléments de $\{1, \ldots ,m\}$,

– $\mathcal{I}$ est l'ensemble des instructions notées et définies, pour tous les entiers naturels i,j,k, par :

$$
\begin{array}{lcl}
[i,constante,j,k] & = & \{(r,s') \in C^2 \mid s_0 = r_0{+}1,\ s_j = k \text{ et } s_i = r_i \text{ ailleur}\},\\
[i,plus,j,k] & = & \{(r,s') \in C^2 \mid s_0 = r_0{+}1,\ s_j = r_j{+}r_k \text{ et } s_i = r_i \text{ ailleur}\},\\
[i,moins,j,k] & = & \{(r,s') \in C^2 \mid s_0 = r_0{+}1,\ s_j = r_j \dot{-} r_k \text{ et } s_i = r_i \text{ ailleur}\},\\
[i,deindirect,j,k] & = & \{(r,s') \in C^2 \mid s_0 = r_0{+}1,\ s_j = r_{r_k} \text{ et } s_i = r_i \text{ ailleur}\},\\
[i,aindirect,j,k] & = & \{(r,s') \in C^2 \mid s_0 = r_0{+}1,\ s_{r_j} = r_k \text{ et } s_i = r_i \text{ ailleur}\},\\
[i,sizero,j,k] & = & \{(r,s') \in C^2 \mid\ s_0 = \begin{cases} r_k, & \text{si } r_j = 0,\\ r_0{+}1, & \text{si } r_j \neq 0 \end{cases} \text{ et } s_i = r_i \text{ ailleur}\},
\end{array}
$$

avec s' ne différent de s que par le fait que $s'_0 = s_0 + 1$ et avec $r_j \dot{-} r_k = \max\{0, r_j - r_k\}$.

4 Programmes et codages universels

4.1 Introduction

Donnons nous une machine $\mathcal{M} = (\Sigma,C,\alpha,\omega,\mathcal{I})$. Codons tout programme P pour $\mathcal{M}$ par un élément de $\Sigma^\star$, noté $code(P)$.

Définition 12 Le couple $(U,code)$, ainsi que le programme U et le codage $code$, sont dits *universels*, si, pour tout programme P et pour tout $x \in \Sigma^\star$,

$$\boxed{out(U, code(P)\cdot x) = out(P,x)}. \tag{1}$$

Si dans la formule précédente on remplace P par U, et x par x par $code(U)^n \cdot x$ on obtient :

$$out(U, code(U)^{n+1}\cdot x) = out(U, code(U)^n \cdot x)$$

et donc :

Propriété 3 *Si $(U,code)$ est un couple universel alors, quel que soit $n \geq 0$ et $x \in \Sigma^\star$,*

$$\boxed{out(U, code(U)^n \cdot x) = out(U,x)}. \tag{2}$$

4.2 Complexité et coefficient d'introspection

Donnons nous une machine $\mathcal{M} = (\Sigma,C,\alpha,\omega,\mathcal{I})$ et un couple universel $(U,code)$ pour $\mathcal{M}$. La *complexité* de ce couple est alors le nombre moyen de transitions effectuées par U pour produire le même effet qu'un transition du programme P figurant comme donné de U. Plus précisément :

Définition 13 Pour un programme donné P et un mot donné x sur Σ tel que $out(P,x) \neq \nearrow$, la *complexité* du couple universel $(U,code)$ est le réel

$$\boxed{\lambda(P,x) = \frac{cout(U, code(P)\cdot x)}{cout(P,x)}.}$$

L'inconvient de cette définition est de faire dépendre la complexité de $(U,code)$ des données de U. Pour une complexité intrinsèque de $(U,code)$, ne dépendant pas de ces données, on introduit le *coefficient d'introspection* de $(U,code)$ dont la définition qui suit est justifiée par la propriété 2 :

Définition 14 Si pour tout $x \in \Sigma^\star$, avec $cout(U,x) \neq \infty$, le réel

$$\boxed{\lim_{n\to\infty} \lambda(U, code(U)^n \cdot x) = \lim_{n\to\infty} \frac{cout(U, code(U)^{n+1}\cdot x)}{cout(U, code(U)^n \cdot x)}}$$

existe et ne dépend pas de x, alors ce réel est le *coefficient d'introspection* du couple universel $(U,code)$.

4.3 Conservation de la complexité et du coefficient d'introspection

Soient $\mathcal{M}_1$ et $\mathcal{M}_2$ deux machines de même alphabet Σ, soient $\mathcal{P}_1$ et $\mathcal{P}_2$ l'ensemble de leurs programmes, soient deux transformations de programmes $f_1 : \mathcal{P}_1 \to \mathcal{P}_2$ et $f_2 : \mathcal{P}_2 \to \mathcal{P}_1$ et soit $(U_2,code_2)$ un couple universel pour $\mathcal{M}_2$. Faisons l'hypothèse :

Hypothèse 1 *La transformation f_1 simule $\mathcal{M}_1$ sur $\mathcal{M}_2$, la transformation f_2 simule $\mathcal{M}_2$ sur $\mathcal{M}_1$ et $code_2 \circ f_1 \circ f_2 = code_2$.*

Du théorème 1 de la page 7 on déduit :

Propriété 4 *Dans le cas particulier où $\mathcal{M}_1$ est une machine de Turing et $\mathcal{M}_2$ une machine de Turing avec direction interne, l'hypothèse 1 est satisfaite en prenant les transformations f_1 et f_2 du théorème 1 et en prenant $code_2$ de la forme $code'_2 \circ f_1 \circ f_2$, avec $code'_2$ une application de type $\mathcal{P}_2 \to \Sigma^\star$.*

Pour la suite, désignons respectivement par x,P_1,P_2 un élément quelconque de $\Sigma^\star,\mathcal{P}_1,\mathcal{P}_2$. On a la propriété suivante :

Propriété 5 *avec l'hypothèse 1 et si $P_1 = f_2(P_2)$ ou $P_2 = f_1(P_1)$, le couple $(U_1, code_1)$, avec $U_1 = f_2(U_2)$ et $code_1 = code_2 \circ f_1$,*

1. *est universel pour $\mathcal{M}_1$,*
2. *a une complexité $\lambda_1(P_1,x)$ non définie ou égale à la complexité $\lambda_2(P_2,x)$ du couple $(U_2,code_2)$, suivant que le réel $\lambda_2(P_2,x)$ n'est pas défini ou est défini,*[3]
3. *admet le même coefficient d'introspection que $(U_2,code_2)$ ou n'en admet pas, suivant que $(U_2,code_2)$ en admet ou n'en admet pas,*
4. *est tel que $code_1(P_1) = code_2(P_2)$.*

Preuve Montrons tout d'abords le point 4. Si $P_2 = f_1(P_1)$, du fait que $code1 = code_2 \circ f_1$, on a

$$code_1(P_1) = code_2(f_1(P_1)) = code(P_2).$$

Si $P_1 = f_2(P_2)$, du fait que $code_2 = code_2 \circ f_1 \circ f_2$ et $code_2 \circ f_1 = code_1$, on a $code_2 = code_1 \circ f_2$ et donc

$$code_2(P_2) = code_1(f_2(P_2)) = code_1(P_1).$$

On montre le point 1 par les égalités qui suivent, où Q_1 est un élément quelconque de $\mathcal{P}_1$:

$out_{\mathcal{M}_1}(Q_1,x) \ =$	(car f_1 simule $\mathcal{M}_1$ sur $\mathcal{M}_2$)
$out_{\mathcal{M}_2}(f_1(Q_1),x) \ =$	(car $(U_2,code_2)$ est universel pour $\mathcal{M}_2$)
$out_{\mathcal{M}_2}(U_2, code_2(f_1(Q_1))\cdot x) \ =$	(car f_2 simule $\mathcal{M}_2$ sur $\mathcal{M}_1$)
$out_{\mathcal{M}_1}(f_2(U_2), code_2(f_1(Q_1))\cdot x) \ =$	(car $f_2(U_2) = U_1$ et $code_2 \circ f_1 = code_1$)
$out_{\mathcal{M}_1}(U_1, code_1(Q_1)\cdot x)$.	

3. Bien entendu $\lambda_i(P_i,x) = \frac{cout_{\mathcal{M}_i}(U_i,\, code_i(P_i)\cdot x)}{cout_{\mathcal{M}_i}(P_i,\, x)}$.

Pour montrer le point 2 il suffit de montrer qu'on a l'égalité de couples :

$$\begin{bmatrix} cout_{\mathcal{M}_1}(U_1, code_1(P_1) \cdot x) \\ cout_{\mathcal{M}_1}(P_1, x) \end{bmatrix} = \begin{bmatrix} cout_{\mathcal{M}_2}(U_2, code_2(P_2) \cdot x) \\ cout_{\mathcal{M}_2}(P_2, x) \end{bmatrix}. \tag{3}$$

Du fait que $U_1 = f_2(U_2)$ et $code_1(P_1) = code_2(P_2)$, puis du fait que f_2 simule $\mathcal{M}_2$ sur $\mathcal{M}_1$ on a :

$$cout_{\mathcal{M}_1}(U_1,code_1(P_1)\cdot x) = cout_{\mathcal{M}_1}(f_2(U_2),code_2(P_2)\cdot x) = cout_{\mathcal{M}_2}(U_2,code_2(P_2)\cdot x)$$

Du fait que f_2 simule $\mathcal{M}_2$ sur $\mathcal{M}_1$, si $P_1 = f_2(P_2)$, et du fait que f_2 simule $\mathcal{M}_1$ sur $\mathcal{M}_2$, si $P_1 = f_2(P_2)$, on a

$$cout_{\mathcal{M}_1}(P_1,x) = cout_{\mathcal{M}_2}(P_2,x).$$

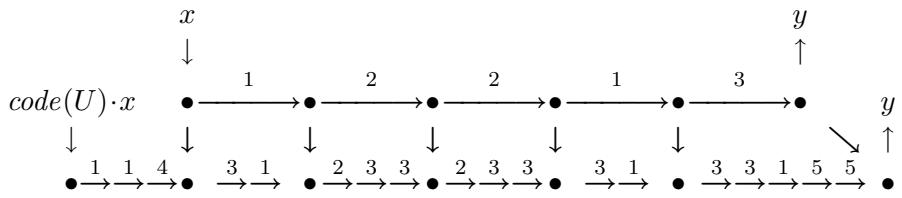
Enfin on montre le point 3 par la suite d'égalités :

$$\begin{aligned}
&\begin{bmatrix} cout_{\mathcal{M}_1}(U_1, code_1(U_1) \cdot code_1(U_1)^n \cdot x) \\ cout_{\mathcal{M}_1}(U_1, code_1(U_1)^n \cdot x) \end{bmatrix} &=& \text{ (en remplaçant } P_1 \text{ par } U_1 \text{ et} \\
&&& \quad x \text{ par } code_1(U_1)^n \cdot x \text{ dans (3))} \\
&\begin{bmatrix} cout_{\mathcal{M}_2}(U_2, code_2(U_2) \cdot code_1(U_1)^n \cdot x) \\ cout_{\mathcal{M}_1}(U_2, code_1(U_1)^n \cdot x) \end{bmatrix} &=& \text{ (car } code_1(U_1) = code_2(U_2)\text{)} \\
&\begin{bmatrix} cout_{\mathcal{M}_2}(U_2, code_2(U_2) \cdot code_2(U_2)^n \cdot x) \\ cout_{\mathcal{M}_2}(U_2, code_2(U_2)^n \cdot x) \end{bmatrix}.
\end{aligned}$$

Ceci termine la preuve.

4.4 Existence et valeur du coefficient d'introspection

On se donne un couple universel $(U,code)$ pour une machine $\mathcal{M} = (\Sigma,C,\alpha,\omega,\mathcal{I})$. Etant donné un mot x sur Σ, on suppose que le calcul du mot y par $y = out(U, x)$ puisse être synchronisé avec le calcul du même mot y par $y = out(U, code(U) \cdot x)$, à la façon du schéma suivant :



Plus précisément on fait l'hypothèse suivante :

Hypothèse 2 *Il existe*

- une fonction de synchronisation $\Phi : C_U \to C_U$, *avec* $\Phi(c)$ *final pour* U *quand* c *est final pour* U,
- une fonction d'étiquetage $\mu : (\bigcup U)^\star \to (1..n)^\star$ *avec* n *un entier positif et* μ *surjective, telle que* $\mu(t_1 \dots t_k) = \mu(t_1) \dots \mu(t_k)$, *pour tout* $t_1 \dots t_k \in (\bigcup U)^\star$,
- une suite initiale d'étiquettes $\delta \in (1..n)^\star$, *indépendante de* x, *telle que*

$$\delta = \mu(trace(U, \alpha(code(U) \cdot x), \Phi(\alpha(x)))), \text{ pour tout } x \in \Sigma^\star,$$

- une réécriture d'étiquettes $\varphi : 1..n \to (1..n)^\star$ *telle que, pour tout* $(c,c') \in \bigcup U$,

$$\varphi(\mu(c,c')) = \mu(trace(U, \Phi(c), \Phi(c'))).$$

On introduit alors le vecteur colonne B et la matrice A :

$$\begin{aligned} B &= \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}, \qquad b_i = \text{nombre d'occurrences de } i \text{ dans } \delta, \\ A &= \begin{bmatrix} a_{11} \cdots a_{nn} \\ \vdots \qquad \vdots \\ a_{1n} \cdots a_{nn} \end{bmatrix}, \quad a_{ij} = \text{nombre d'occurrences de } i \text{ dans } \varphi(j). \end{aligned} \tag{4}$$

et on conclut par le théorème principal du papier :

Théorème 2 *Si la matrice* A *admet une valeur propre réelle* λ, *dont l'ordre de multiplicité est égal à* 1 *et dont la valeur est strictement supérieure à* 1 *et strictement supérieure au plus grand module* λ' *des autres valeurs propres de* A *alors :*

- *la matrice limite* $\bar{A} = \lim_{n\to\infty}(\frac{1}{\lambda}A)^n$ *existe, et n'est pas partout nulle,*
- *si* $||\bar{A}B|| \neq 0$, *le coefficient d'introspection de* U *existe et vaut* λ,
- *si* ℓ *un réel tel que* $\lambda' < \ell < \lambda$ *et les* X_i *des vecteurs colonnes définies par* $X_0 = B$ *et* $X_{n+1} = \frac{1}{\ell}AX_n$, *alors, quand* $n \to \infty$,

$$||X_n|| \to \begin{cases} 0, & \text{si } ||\bar{A}B|| = 0, \\ \infty, & \text{si } ||\bar{A}B|| \neq 0. \end{cases}$$

Ici $||X||$ désigne la somme des composantes de X.

5 Nos couples universels pour nos deux types de machines de Turing

5.1 Introduction

Nous présentons ici deux couples universels, particulièrement efficaces, $(U_1, code_1)$ et $(U_2, code_2)$, l'un pour la machine de Turing $\mathcal{M}_1$ d'alphabet $\Sigma = \{\mathtt{o},\mathtt{i},\mathtt{z}\}$ et l'autre pour la machine de Turing avec direction interne $\mathcal{M}_2$ de même alphabet Σ.

Le couple $(U_1, code_1)$ est construit à partir du couple $(U_2, code_2)$ en prenant $U_1 = f_2(U_2)$ et $code_1 = code_2 \circ f_1$, où f_1 et f_2 sont les transformations de programmes introduites dans le

théorème 1 à la page 7. Du fait que $code_2$ est de la forme $code'_2 \circ f_1 \circ f_2$, d'après les propriétés 4 et 5, le couple $(U_1,code_1)$ est donc bien universel pour $\mathcal{M}_1$ et les propriétés de complexité du couple $(U_2,code_2)$ lui sont transposables.

Signalons que U_1 comporte 361 instructions et 106 états alors que U_1 ne comportent que 184 instructions et 54 états. Il reste à présenter le couple $(U_2,code_2)$.

5.2 Fonction de codage du couple universel $(U_2,code_2)$

Pour affecter une position à chaque instruction $[q_i,abs,q_j]$ d'un programme quelconque pour $\mathcal{M}_2$, introduisons tout d'abords les nombres :

$$\pi(i,a) = 4(i-1) + \begin{cases} 1, & \text{si } a = \mathtt{u} \\ 2, & \text{si } a = \mathtt{o} \\ 3, & \text{si } a = \mathtt{i} \\ 4, & \text{si } a = \mathtt{z} \end{cases}, \qquad \pi(i) = \tfrac{1}{2}(f(i,\mathtt{o}) + f(i,\mathtt{i})).$$

définis pour tout entier positif i et tout symbole $a \in \{\mathtt{u},\mathtt{o},\mathtt{i},\mathtt{z}\}$. Soit alors un programme P_2 pour $\mathcal{M}_2$ et soit le programme $P'_2 = f_1(f_2(P_2))$. On prend $code_2(P_2) = code'_2(P'_2)$, avec $code'_2(P'_2)$ le mot sur $\{\mathtt{o},\mathtt{i},\mathtt{z}\}$:

$$\mathtt{z}I_{4n}\mathtt{z}\ldots\mathtt{z}I_{k+1}\mathtt{z}I_k\mathtt{z}I_{k-1}\mathtt{z}\ldots\mathtt{z}I_1\mathtt{zoi}\ldots\mathtt{izz},$$

où n est le nombre d'états de P'_2, où la taille de la *navette* $\mathtt{oi}\ldots\mathtt{iz}$ est égale à la plus longue taille des I_k moins 5, et où, pour tout $a \in \Sigma_{\mathtt{u}}$ et $i \in 1..n$,

$$I_{\pi(i,a)} = \begin{cases} \overline{[q_i,abs,q_j]}, & \text{s'il existe } b,s,j \text{ avec } [q_i,abs,q_j] \in P'_2, \\ \mathtt{oi}, & \text{sinon,} \end{cases}$$

avec,

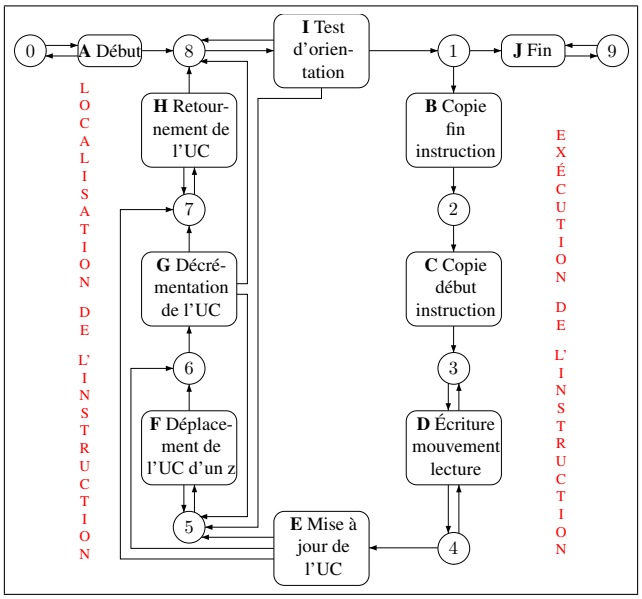
$$\overline{[q_i,a,b,s,q_j]} = \begin{cases} \mathtt{i}a_m\ldots a_2\mathtt{o}, & \text{si } \pi(j) - h(i,a) > 0, \\ \mathtt{o}a_2\ldots a_m\mathtt{i}, & \text{si } \pi(j) - h(i,a) < 0, \end{cases}$$

avec a_2a_3 égal à $\mathtt{io}$, $\mathtt{oi}$, $\mathtt{ii}$, suivant que b est égal à $\mathtt{u}$, $\mathtt{o}$, $\mathtt{i}$, $\mathtt{z}$, avec $a_4 = \mathtt{o}$ ou $a_4 = \mathtt{i}$ suivant que $s = +$ ou $s = -$ et avec $\mathtt{i}a_m\ldots a_5$ un nombre binaire ($\mathtt{o}$ pour 0 et $\mathtt{i}$ pour 1) dont la valeur est égale à $|\pi(j) - \pi(i,a)| + \frac{3}{2}$.

5.3 Fonctionnement du couple universel $(U_2,code_2)$

Ainsi que nous l'avons déjà mentionné, le programme U_2 est composé de 54 états $q_1,\ldots,q_{54}$ et de 184 instructions. Ces instructions sont regroupés en 10 modules $A,B,C,\ldots,J$ organisés

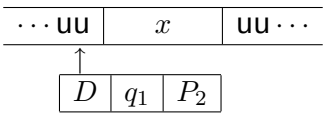
comme suit :



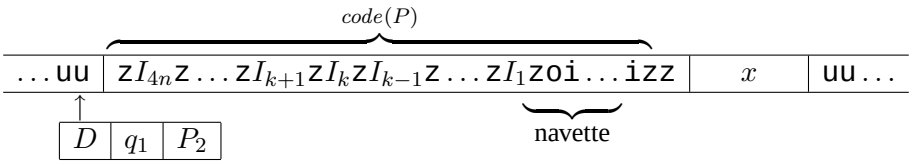
Les nombres 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 désignent respectivement les états q_1, q_{24}, q_{35}, q_{43}, q_{49}, q_{15}, q_{13}, q_7, q_{10}, q_{23}. Le programme U_2 au complet est donné en appendice sous forme d'un graphe dont les sommets sont les états et les arrêtes les instructions de U_2 : chaque instruction $[q_i,abs,q_j]$ est représentée par une flèche, étiquetée abs, allant de q_i à q_j. A noter que les sommets a, b, c et 7 ont deux occurrences qui doivent être fusionnées.

Configurations initiales

Au départ la machine exécutant P_2 est dans la configuration initiale



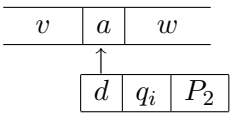
et la machine exécutant U_2 est dans la *configuration intiale correspondante*



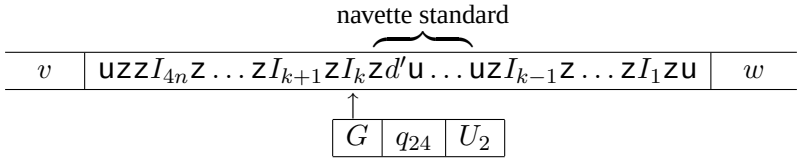
Alors que P_2 n'effectue aucune transition U_2 exécute une suite de transitions d'initialisations, toujours la même, faisant intervenir les instructions du module A et quelques instructions déjà présentes dans les modules I,H,G,F. Les machines exécutant P_2 et U_2 se retrouvent alors respectivement dans les configurations courantes qui suivent avec $k = 1$.

Configurations courantes

Alors que la machine exécutant P_2 est dans la configuration courante

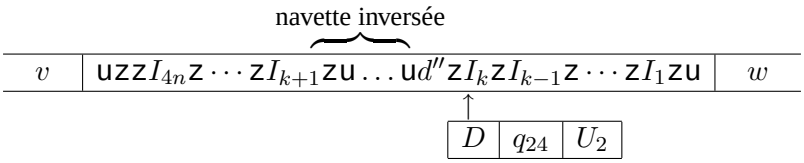


la machine exécutant U_2 est dans la *configuration courante correspondante*



(5)

ou



(6)

suivant que I_k, avec $k = \pi(i,a)$, est de la forme i$a_m \dots a_2$o dite *standard*, ou de la forme i$a_m \dots a_2$o, dite *inverséee*. La tête d'écriture-lecture est positionnée sur a_3 ou sur le z qui suit I_k quand I_k est l'instruction vide oi. Suivant que d est égal à G ou D, le symbole d' est égale à u ou o, si I_k est standard, et à o ou u, si I_k est inversé.

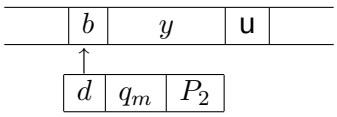
Tant que la configuration courante de P_2 n'est pas finale, P_2 effectue une transition pour passer à la configuration courante suivante et U_2 effectue une suite de transitions pour passer à la prochaine configuration courante correspondante. Plus précisément, à partir de l'information contenue dans I_k, le programme U_2

- met à jour la direction interne contenue dans la navette (module B),
- transfert dans la navette un nombre binaire servant de base pour calculer le nombre d'instruction à sauter vers la gauche ou vers la droite (suivant que la navette est sous forme standard ou inverséee) (module C),
- simule l'écriture d'un symbole, le mouvement du ruban, puis la lecture d'un nouveau symbole(module D).
- en fonction du symbole lu et du nombre binaire contenu dans la navette calcul le nombre exact d'instruction à sauter pour se positionner sur la prochaine instruction à exécuter (module E),
- déplace la navette en la retournant éventuellemnt pour la positionner correctement contre la prochaine instruction à exécuter.

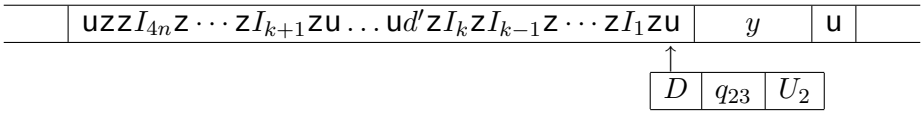
Lorsque la configuration courante de P_2 devient finale, la configuration courante correspondante de U_2 est de la forme (6) avec I_k égal à l'instruction vide oi. Le programme U_2 passe alors par une suite de transitions (module J) pour atteindre la configuration finale correspondante. Les machines exécutant P_2 et U_2 se retrouvent alors dans les configurations finales qui suivent.

Configurations finales

Alors que la machine exécutant P_2 termine dans la configuration finale



la machine exécutant U_2 termine dans la *configuration finale correspondante*



avec $I_k = \mathtt{oi}$, $k = \pi(m,b)$ et d' égal à o ou u, suivant que d est égal à G ou D.

6 Complexité et coefficient d'introspection de nos deux couples

6.1 Complexité générale

Soit P_2 un programme quelconque pour $\mathcal{M}_2$, la machine de Turing avec direction interne d'alphabet $\Sigma = \{\mathtt{o},\mathtt{i},\mathtt{z}\}$. De la façon dont nous avons défini la fonction de codage $code_2$, pour $\ell = 2$, on a

$$\boxed{|code_\ell(P_\ell)| = \mathcal{O}(n \log n)},$$

où n est le nombre d'instructions de $propre(P_\ell)$. Ce résultat est aussi valable pour $\ell = 1$, avec P_1 un programme quelconque pour $\mathcal{M}_1$, la machine de Turing classique de même alphabet que $\mathcal{M}_2$. Ceci est du au fait que $code_1(P_1) = code_2(f_1(P_1))$ et que le nombre d'instructions de $propre(f_1(P_1))$ est au plus égal au double de celui de P_1.

En reprenant la description du fonctionnement de U_2 à la section 5.3, on conclut que si P_2 effectue h transitions alors il existe des réels positifs k_i ne dépendant ni de h ni de n tels que U_2 effectue au plus :

- $k_1 \log^2 n$ transitions pour atteindre la première configuration correspondant à une configuration courante de P_2,
- $hk_1 \log^2 n$ transitions pour transférer de l'information d'une instruction I_i dans la navette qui la cotoye,
- $hk_2 n \log n$ transitions pour simuler la lecture d'un symbole, le mouvement de la tête et la lecture d'un symbole,
- $hk_3 n \log^2 n$ transitions pour déplacer la navette,
- $hk_4 n \log^2 n$ pour atteindre une configuration finale à prtir d'une configuration correpondant à une configuration finale de P_2,

et donc au total au plus $hk_5 n \log^2 n$ instructions.

Donc pour $\ell = 2$, il existe un réel positif k indépendant de $x \in \Sigma^\star$ et de n tel que

$$\boxed{\lambda_\ell(P_\ell,x) \leq kn \log^2 n}$$

Si au lieu de mesurer la complexité en fonction de n on le fait en fonction de $m = |code_\ell(P_\ell)|$, on conclut qu'il existe un réel positif k, indépendant de $x \in \Sigma^\star$ et de m, tel que

$$\boxed{\lambda_\ell(P_\ell,x) \leq km \log m}$$

Ces deux résultats sont aussi valables pour $\ell = 1$, avec P_1 un programme quelconque pour $\mathcal{M}_1$.

6.2 Complexité sur des exemples

Sur des exemples nous avons obtenus les résutats de complexité suivants pour les couples $(U_\ell,code_\ell)$ avec $\ell = 1$ et $\ell = 2$.

x	$cout($ $P_\ell,x)$	$cout(U_\ell,$ $code_\ell(P_\ell)\cdot x)$	$cout(U_\ell,code_\ell(U_\ell)\cdot$ $code_\ell(P_\ell)\cdot x)$	$\lambda_\ell(P_\ell,x)$	$\lambda_\ell(U_\ell,$ $code_\ell(P_\ell)\cdot x)$
ε	2	5 927	22 974 203	2 963,50	3 876,19
`o`	6	13 335	51 436 123	2 222,50	3 857,23
`oi`	12	23 095	88 887 191	1 924,58	3 848,76
`oiz`	20	35 377	136 067 693	1 768,85	3 846,22
`oizo`	30	49 663	190 667 285	1 655,43	3 839,22

Ici P_ℓ, avec $P_1 = f_2(P_2)$, est un programme d'inversion tel que pour tout $n \geq$ on ait $out(P_\ell, a_1a_2 \dots a_n) = a_n \dots a_2a_1$, avec les a_i pris dans $\{$`o`,`i`,`z`$\}$. Dans les deux cas, $\ell = 1$ et $\ell = 2$, le programme P_ℓ comporte 32 instructions et 9 états. On a $|code_\ell(P_\ell)| = 265$ et $|code_\ell(U_\ell)| = 1552$.

6.3 Coefficient d'introspection

Pour satisfaire l'hypothèse 2, $code_2(U_2) = f_1(f_2(code_2(U_2)))$ et la fonction d'étiquetage μ est définie de façon à ce que, pour toutes transitions (c_1,c_2) et $(c'_1,c_2\prime)$ de $\bigcup U_2$, les entiers $\mu(c_1,c_2)$ et $\mu(c'_1,c_2\prime)$ soient égaux si et seulement si à la fois :

- les états de c_1 et c'_1 sont égaux,
- les symboles sur lesquels la tête d'écriture-lecture est positionné dans c_1 et c'_1 sont égaux,
- les symboles sur lesquels la tête d'écriture-lecture sont positionné dans c_2 et c'_2 sont égaux,
- les directions dans c_2 et $c_2\prime$ sont les mêmes.

La fonction Φ est définie, pour toute configuration de U_2, avec $P_2 = U_2$, par,

$$\Phi(c) = \begin{cases} \text{configuration courante correspondante de } P_2, \text{ si } c \text{ n'est pas finale pour } P_2, \\ \text{configurations finale correspondante de } P_2, \text{ si } c \text{ est finale pour } P_2. \end{cases}$$

Après avoir calculé le vecteur B et la matrice A on a vérifié par la proprité 2 que $(U_2,code_2)$ admetait bien un coefficient d'introspection et calculé sa valeur : pour $\ell = 2$ et tout mot x sur Σ tel que $cout(U_\ell, x) \neq \infty$,

$$\boxed{\lim_{n\to\infty} \frac{cout(U_\ell,\, code_\ell(U_\ell)^{n+1}\cdot x)}{cout(U_\ell,\, code(U_\ell)^n\cdot x)} = 3672{,}98.}$$

Ce resultat est aussi valable pour $\ell = 1$.

7 Notre couple universel pour machine arithmétique

Il est intéressant de comparer la complexité de notre programme universel U_1 pour machine de Turing avec celle d'un programme universel U_3 pour machine arithmétique à adressage indirect de même alphabet $\Sigma = \{\texttt{o},\texttt{i},\texttt{z}\}$.

Nous avons donc écrit un tel programme universel U_3 en 103 instructions. De la façon dont notre couple universel $(U_3, code_3)$ fonctionne nous avons pu montré que :

Propriété 6 *Il existe un nombre positif k tel que, quel que soit le programme P_3 et le mot x sur Σ, avec $cout(P_3, x) \neq \infty$,*

$$\boxed{\lambda_3(P_3, x) = \frac{cout(U_3, code_3(P_3) \cdot x)}{cout(P_3, x)} \leq 35 + k\frac{|code(P)|}{cout(P_3, x)}}$$

ce qui montre que la taille du programme P_3 devient négligeable dés qu'on effectue un grand nombre de transition. Sur des exemples particuliers nous avons obtenus les résutats suivants

x	$cout($ P_3,$x)$	$cout(U_3,$ $code_3(P_3)\cdot x)$	$cout(U_3,code_3(U_3)\cdot$ $code_3(P_3)\cdot x)$	$\lambda_3(P_3,x)$	$\lambda_3(U_3,$ $code_3(P_3)\cdot x)$
ε	12	2 372	72 110	197,66	30,40
o	16	2 473	74 758	154,56	30,23
oi	31	2 860	84 916	92,26	29,69
oiz	35	2 961	87 564	84,60	29,57
oizo	50	3 348	97 722	66,96	29,19

où P_3 est un programme d'inversion de 21 instructions tel que, pour tout $n \geq$ on ait $out(P_3, a_1a_2 \ldots a_n) = a_n \ldots a_2a_1$, avec les a_i pris dans $\{\texttt{o},\texttt{i},\texttt{z}\}$. Signalons que $|code_3(P_3)| = 216$ et $|code(U_3)| = 1042$.

Comme coefficient d'introspection nous avons obtenu :

$$\boxed{\lim_{n\to\infty} \frac{cout(U_3, code(U_3)^{n+1}\cdot x)}{cout(U_3, code(U_3)^n \cdot x)} = 26{,}27228613}$$

8 Conclusion

A moins de « tricher », il est difficile d'améliorer le coefficient d'introspection de notre programme universel pour machine de Turing auquel nous avons consacré tant de temps. Supposons, comme c'est le cas, que nous disposions d'un premier couple universal $(U, code)$ pour une machine de Turing.

Une première façon de tricher consiste, à partir de notre couple universel $(U, code)$, de construire le couple universel $(U, code')$ avec

$$code'(P) = \begin{cases} \varepsilon, & \text{si } P = U, \\ code(P), & \text{si } P \neq U. \end{cases}$$

On a alors

$$\frac{cout(U', code(U')^{n+1}\cdot x)}{cout(U', code(U')^n\cdot x)} = \frac{cout(U', x)}{cout(U', x)} = 1$$

et $(U, code')$ est un couple universel dont le coefficient d'introspection vaudrait 1.

Une deuxième façon de tricher, plus subtile, ne modifie pas la fonction de codage $code$. A partir du programme universel U on construit un programme U' qui, après avoir effacé autant de fois que possible un certain mot z du début de sa donnée, se comporte comme U. D'après le théorème de récursion [2, 4], il est possible de choisir z égal à $code(U')$ et d'obtenir ainsi un programme universel U' tel que, pour tout $y \in \Sigma^\star$ n'ayant pas $code(U)'$ pour préfixe,

$$cout(U', code(U')^n \cdot y) = nk_1 + k_2(y), \tag{7}$$

où k_1 et $k_2(y)$) sont des entiers strictement positif, avec k_1 ne dépendant pas de y. On a alors :

$$\frac{cout(U', code(U')^{n+1} \cdot y)}{cout(U', code(U')^n \cdot y)} = \frac{cout(U,x)+(n+1)k_1+k_2(y)}{cout(U,x)+nk_1+k_2(y)} =$$

$$1 + \frac{k_1}{cout(U,x)+k_2(y)+nk_1}.$$

En faisant tendre n vers l'infini on obtient ainsi un coefficient d'introspection égal à 1 pour le couple $(U', code)$.

Malheureusement notre définition du coefficient d'introspection à la page 8 n'interdit pas ces deux types de tricheries. Si on impose l'hypothèse 2, la première tricherie est toujours possible et la deuxième peut être empéchée en interdisant à la function $\varphi(t)$ de produire la suite vide. Mais cette interdiction semble ad hoc.

Ce qu'on voudrait vraiment interdire c'est que la fonction $code$ ou le programme U « traite différemment » le programme P suivant qu'il est égal ou différent de U. Exprimer cette interdiction par une définition plus restrictive du coefficient d'introspection est un problème ouvert.

Références

[1] Marvin Minsky, *Computations: Finite and Infinite Machines*, Prentice-Hall, 1967.

[2] Hartley Roger, *Theory of Recursive Functions and Effective Computability*, McGraw-Hill, 1967, also MIT Press, fifth printing, 2002.

[3] Yurii Rogozin, Small universal Turing machines, *Theoretical Computer Science*, Volume 168, nunéro 2, novembre 1996.

[4] Michael Sipser, *Introduction to the Theory of Computation*, PWS Publishing Company, 1997.

Appendice : graphe du programme universel U_2