DM 1 – à rendre le 3 octobre

Le soin accordé à la rédaction sera pris en compte dans l'évaluation de la copie.

Exercice 1.
Justifier de manière détaillée que la fonction qui à n associe le $(n+1)$-ème nombre pair s'écrivant comme somme de deux nombres premiers est récursive primitive.

Solution de l'exercice 1.

Dans ce qui suit, on donne une réponse détaillée, c'est-à-dire que l'on revient aux définitions précises des opérations sur les fonctions récursives primitives qui permettent d'en obtenir de nouvelles afin de montrer que la fonction obtenue est bien récursive primitive.

Rappelons que la fonction $\pi : \mathbb{N} \to \mathbb{N}$ qui à n associe le $n+1$-ème nombre premier est récursive primitive (cours).

Montrons que l'ensemble A des entiers pairs sommes de deux nombres premiers est récursif primitif. On a par définition $x \in A$ ssi $\exists a, b, y : x = 2y$ et $x = \pi(a) + \pi(b)$. Remarquons que comme $\pi(n) > n$ pour tout n et comme $x = 2y \Rightarrow y \leqslant x$, l'ensemble A est en fait défini par quantification bornée.

Plus précisément, sion considère l'ensemble $\tilde{A}$ des quadruplets (x, a, b, y) tels que $x = 2y$ et $x = 2y = \pi(a)+\pi(b)$, alors $\tilde{A}$ est récursif primitif, et donc on peut appliquer trois fois un schéma de quantification bornée pour obtenir que l'ensemble

$$B = \{(x, x_1, x_2, x_3) : \exists a \leqslant x_1 \exists b \leqslant x_2 \exists y \leqslant x_3 : (x, a, b, y) \in A\}$$

est récursif primitif, et alors on voit que $x \in A$ ssi $(x, x, x, x) \in B$ donc A est récursif primitif par composition.

Notons f la fonction qui à n associe le $(n+1)$-ème nombre pair s'écrivant comme somme de deux nombres premiers. Comme le seule nombre premier pair est 2, une borne explicite pour le $n+1$-ème nombre pair s'écrivant comme est alors donnée par $\pi(n+1)+3$. Alors f est définie par récurrence par :

— $f(0) = 4$
— $f(n+1) = \mu k \leqslant \pi(n+1)+3 : k > f(n)$ et $k \in A$

Pour montrer que f est bien récursive primitive, il suffit donc de montrer que la fonction $h : (n, y) \mapsto \mu k \leqslant \pi(n+1)+3 : k > y$ et $k \in A$ est récursif primitive. Introduisons donc la fonction $h' : (y, z) \mapsto \mu k \leqslant z : k > y$ et $k \in A$. Comme l'ensemble des (k, y, z) tels que $k > y$ et $k \in A$ est récursif primitif, la fonction h' est récursive primitive par schéma μ-borné. Ainsi comme $h(n, y) = h'(y, \pi(n+1)+3)$, la fonction h est récursive primitive. Comme expliqué plus haut, ceci conclut la preuve que la fonction f est récursive primitive.

Ainsi la fonction qui à n associe le $(n+1)$-ème nombre pair s'écrivant comme somme de deux nombres premiers est récursive primitive.

Exercice 2.
Soit $\mathcal{C}$ le plus petit sous-ensemble de $\mathcal{F}$ qui contient les projections, les fonctions constantes, la fonction successeur, est stable par composition et par *itération*, c'est-à-dire que si $f_1, ..., f_p \in \mathcal{C} : \mathbb{N}^p \to \mathbb{N}$ alors les fonction $g_1, ..., g_p : \mathbb{N}^{p+1} \to \mathbb{N}$ définies par

$$\begin{cases} g_i(x_1, ..., x_p, 0) &= x_i \\ g_i(x_1, ..., x_p, y+1) &= f_i(g_1(x_1, ..., x_p, y), ..., g_p(x_1, ..., x_p, y)) \end{cases}$$

sont dans $\mathcal{C}$. Montrer que $\mathcal{C}$ est égal à l'ensemble des fonctions récursives primitives.

Solution de l'exercice 2.

Notons $\mathcal{P}$ l'ensemble des fonctions récursives primitives, il s'agit de montrer que $\mathcal{P} = \mathcal{C}$.

Montrons d'abord que $\mathcal{C} \subseteq \mathcal{P}$. Comme $\mathcal{C}$ est le plus petit ensemble de fonctions dans $\mathcal{F}$ contenant les projections, les constantes, la fonction successeur, et stable par composition et itération, il suffit de montrer que $\mathcal{P}$ satisfait les mêmes propriétés. Par définition de $\mathcal{P}$, il suffit donc de voir que $\mathcal{P}$ est stable par itération, ce que l'on va faire au moyen d'un codage de $(g_1, ..., g_p)$.

Plus précisément, soient $f_1, ..., f_p \in \mathcal{P} : \mathbb{N}^p \to \mathbb{N}$ et soient les fonction $g_1, ..., g_p : \mathbb{N}^{p+1} \to \mathbb{N}$ définies par

$$\begin{cases} g_i(x_1, ..., x_p, 0) &= x_i \\ g_i(x_1, ..., x_p, y+1) &= f_i(g_1(x_1, ..., x_p, y), ..., g_p(x_1, ..., x_p, y)) \end{cases}$$

On veut montrer que $g_1, ..., g_p \in \mathcal{P}$. Considérons la fonction

$$g : (x_1, ..., x_p, y) \mapsto \alpha_p(g_1(x_1, ..., x_p, y), ..., g_p(x_1, ..., x_p, y)).$$

Alors on montre par une récurrence immédiate que g est en fait définie par la récurrence suivante, où on note $\bar{x} = (x_1, ..., x_p)$:

$$\begin{aligned} g(\bar{x}, 0) &= \alpha_p(\bar{x}) \\ g(\bar{x}, y+1) &= \alpha_p\left(f_1(\beta_p^1 g(\bar{x}, y), ..., \beta_p^1 g(\bar{x}, y)), ..., f_p(\beta_p^p g(\bar{x}, y), ..., \beta_p^p g(\bar{x}, y))\right) \end{aligned}$$

Ainsi g est récursive primitive, et par construction pour tout $i = 1, ..., p$ on a $g_i = \beta_p^i g$ donc g_i est récursive primitive. Ainsi $\mathcal{P}$ est stable par itération, ce qui conclut la preuve de l'inclusion $\mathcal{P} \subseteq \mathcal{C}$.

Montrons maintenant que $\mathcal{P} \subseteq \mathcal{C}$. Comme $\mathcal{P}$ est le plus petit ensemble de fonctions dans $\mathcal{F}$ contenant les projections, les constantes, la fonction successeur, et stable par composition et récurrence, il suffit de montrer que $\mathcal{C}$ satisfait les mêmes propriétés. Par définition de $\mathcal{C}$, il suffit donc de voir que $\mathcal{C}$ est stable par récurrence, ce que l'on va faire en voyant les récurrences comme un cas particulier d'itération.

Soient $g \in \mathcal{F}_p \cap \mathcal{C}$ et $h \in \mathcal{F}_{p+2} \cap \mathcal{C}$, il s'agit donc de montrer que la fonction $f \in \mathcal{F}_{p+1}$ définie par

$$\begin{aligned} f(x_1, ..., x_p, 0) &= g(x_1, ..., x_p) \\ f(x_1, ..., x_p, y+1) &= h(x_1, ..., x_p, y, f(x_1, ..., x_p, y)) \end{aligned}$$

est bien dans $\mathcal{C}$. On va définir, pour $i \in \{1, ..., p+2\}$, des fonctions g_i de la manière suivante :

— pour $i = 1, ..., p$, on pose $g_i(x_1, ..., x_p, z, y) = x_i$,
— pour $i = p+1$, on pose $g_{p+1}(x_1, ..., x_p, z, y) = y$ et
— pour $i = p+2$, on pose $g_{p+2}(x_1, ..., x_p, z, y) = f(x_1, ..., x_p, y)$.

Alors $(g_1, ..., g_{p+2})$ est définie par itération à partir de fonctions dans $\mathcal{C}$: en effet pour $i = 1, ..., p$, une preuve immédiate par récurrence donne montre que

$$\begin{cases} g_i(x_1, ..., x_p, z, 0) &= x_i \\ g_i(x_1, ..., x_p, z, y+1) &= x_i = g_i(x_1, ..., x_p, y) \end{cases}$$

(la fonction f_i dans la définition de l'itération est donc la projection sur la i-ème coordonnée), pour $i = p+1$ on a

$$\begin{cases} g_{p+1}(x_1, ..., x_p, z, 0) &= y \\ g_{p+1}(x_1, ..., x_p, z, y+1) &= y = S(g_{p+1}(x_1, ..., x_p, z, y)) \end{cases}$$

(la fonction f_{p+1} dans la définition de l'itération est donc le successeur composé projection sur la $p+1$-ème coordonnée) et enfin pour $i = p+2$ on a, en posant $\bar{x} = (x_1, ..., x_p)$:

$$\begin{cases} g_{p+2}(x_1, ..., x_p, z, 0) &= g(x_1, ..., x_p) \\ g_{p+2}(x_1, ..., x_p, z, y+1) &= y = h(g_1(\bar{x}, z, y), ..., g_p(\bar{x}, z, y), g_{p+1}(\bar{x}, y, z), g_{p+2}(\bar{x}, y)) \end{cases}$$

(la fonction f_{p+2} dans la définition de l'itération est donc h). Ainsi la fonction f est bien obtenue par itération à partir de fonctions dans $\mathcal{C}$, ce qui termine la preuve de l'inclusion $\mathcal{C} \subseteq \mathcal{P}$.

On a donc bien $\mathcal{P} = \mathcal{C}$ par double inclusion.

Problème 3.

On va définir une variante des machines RAM, les machines PRAM (Primitive Random Access Memory) et chercher à comprendre ce qu'elles calculent. Une machine PRAM a un nombre $p \in \mathbb{N}$ de registres et prend un nombre $q \in \mathbb{N}$ d'arguments. Elle possède également un accumulateur où les calculs seront effectués.

Une *configuration* d'une machine PRAM à p registres et q arguments est la donnée d'un quadruplet

$$((x_1, ..., x_q), ((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p, a, s)$$

où $x_1, ..., x_q, r_{i,n}, a, s \in \mathbb{N}$ et tous les $r_{i,n}$ sont nuls sauf un nombre fini d'entre eux.

Les x_k sont les arguments, les $r_{i,n}$ sont le contenu des registres, a est le contenu de l'accumulateur (où les calculs sont effectués) et enfin s est le contenu de la sortie. La *configuration initiale* d'une machine PRAM d'arguments $(x_1, ..., x_p)$ est $((x_1, ..., x_p), ((0)_{n\in\mathbb{N}})_{i=1}^p, 0, 0)$.

Une *instruction* est une chaîne de caractères et d'entiers de la forme suivante

— `load#`k (mettre l'entier k dans l'accumulateur)
— `load`$_j k$ (mettre le contenu de $r_{j,k}$ dans l'accumulateur)

— $\texttt{load}_j(k)$ (mettre le contenu de $r_{j,r_{j,k}}$ dans l'accumulateur)
— $\texttt{store}_j k$ (enregistrer le contenu de l'accumulateur dans $r_{j,k}$)
— $\texttt{store}_j(k)$ (enregistrer le contenu de l'accumulateur dans $r_{j,r_{j,k}}$)
— `incr` (incrémente l'accumulateur)
— `decr` (décrémente l'accumulateur ; s'il est nul, ne fait rien)
— `do` (début de boucle effectuée autant de fois que le contenu de l'accumulateur)
— `enddo` (fin de la boucle `do`)
— $\texttt{read}_k$ (mettre la k-ème entrée x_k dans l'accumulateur)
— `write` (mettre le contenu de l'accumulateur dans la sortie)

On définit par induction un *programme* comme une suite d'instructions :
— La suite vide est un programme.
— Si P est un programme, alors P suivi d'une instruction de la forme $\texttt{load}_j\#n$, $\texttt{load}_j n$, $\texttt{load}_j(n)$, $\texttt{store}_j n$, $\texttt{store}_j(n)$, `incr`, `decr`, $\texttt{read}_k$, `write` est un programme.
— Si P et Q sont des programmes, alors P suivi de `do`, puis de Q puis de `enddo` est un programme.

On définit par induction le résultat d'un programme sur une configuration, qui est une nouvelle configuration.
— Le programme vide ne modifie pas la configuration
— Si P est un programme dont le résultat est $((x_1,...,x_p),((r_{j,n})_{n\in\mathbb{N}})_{j=1}^p,a,s)$ et P est suivi de
  — $\texttt{load}\#k$ la configuration résultante est $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,k,s)$
  — $\texttt{load}_j k$ la configuration résultante est $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,r_{j,k},s)$
  — $\texttt{load}_j(k)$ la configuration résultante est $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,r_{j,r_{j,k}},s)$
  — $\texttt{store}_j k$ la configuration résultante est $((x_1,...,x_p),((r'_{i,n})_{n\in\mathbb{N}})_{j=1}^p,a,s)$ avec $r'_{i,n}=r_{i,n}$ sauf pour $i=j$ et $n=k$ auquel cas $r'_{j,k}=a$
  — $\texttt{store}_j(k)$ la configuration résultante est $((x_1,...,x_p),((r'_{i,n})_{n\in\mathbb{N}})_{j=1}^p,a,s)$ avec $r'_{i,n}=r_{i,n}$ sauf pour $i=j$ et $n=r_{j,k}$ auquel cas $r'_{j,r_{j,k}}=a$
  — `incr` la configuration résultante est $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,a+1,s)$
  — `decr` la configuration résultante est $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,a\dot{-}1,s)$
  — $\texttt{read}_k$ la configuration résultante est $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,x_k,s)$
  — `write` la configuration résultante est $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,a,a)$
— Si le programme est de la forme $P,\texttt{do},Q,\texttt{enddo}$, la configuration résultante est obtenue en appliquant une fois le programme P à la configuration $((x_1,...,x_p),((r_{j,n})_{n\in\mathbb{N}})_{j=1}^p,a,s)$ puis, en notant $((x'_1,...,x'_p),(r'_{j,n})_{n\in\mathbb{N}})_{j=1}^p,a',s')$ la configuration obtenue, en itérant a' fois le programme P sur cette dernière (en particulier si $a=0$, on ne modifie pas la configuration obtenue).

Chaque programme P définit une fonction $f_P:\mathbb{N}^p\to\mathbb{N}$ donnée par $f(x_1,...,x_p)=s$ où $((x_1,...,x_p),((r_{i,n})_{n\in\mathbb{N}})_{i=1}^p,a,s)$ est le résultat de l'exécution du programme P sur la configuration initiale d'arguments $(x_1,...,x_p)$. Une telle fonction est dite *PRAM-exécutable*.

1. Montrer que les fonctions projection, successeur et constantes sont PRAM-exécutables (on se contentera d'écrire les programmes correspondants).
2. Montrer que la fonction somme est PRAM-exécutable.
3. Montrer que la fonction $(j,n)\mapsto pn+j+1$ est PRAM-exécutable.
4. Montrer que toute fonction PRAM-exécutable est PRAM-exécutable par une machine PRAM avec un seul registre.
5. Montrer que toute fonction récursive primitive est PRAM-exécutable.
6. On travaille désormais avec un seul registre (ce qui ne change pas l'ensemble des fonctions PRAM-exécutables d'après la question 4). Donner un codage récursif primitif des configurations.
7. Montrer par induction que pour tout programme P, la fonction qui au code d'une configuration C associe le code de la configuration C' résultante du programme P appliqué à C est récursive primitive.
8. Conclure que toute fonction est PRAM-exécutable ssi elle est récursive primitive.

Solution de l'exercice 3.

1. Le code suivant implémente la fonction projection sur la i-ème coordonnée :

```
read_i
write
```

Le code suivant implémente la fonction successeur :

```
read_1
incr
write
```

Le code suivant implémente la fonction constante égale à k :

```
load#k
write
```

2. Voici un code qui implémente la fonction $(x, y) \mapsto x + y$:

```
read_1                  # On lit x
store_1 0               # On stocke x
read_2                  # On lit y
do                      # On va ajouter y fois 1 à x
     load_1 0
     incr
     store_1 0
enddo
write
```

3. Voici un code qui implémente la fonction $(j, n) \mapsto pn + j + 1$:

```
load#p
do                              # On va ajouter p fois n à 0 (= r_{1,0})
     read_2                     # On lit n
     do                         # On ajoute n fois 1 à r_{1,0}
          load_1 0
          incr
          store_1 0
     enddo
enddo                           # Après cette boucle r_{1,0} = pn
read_1                          # On lit j
do                              # On ajoute j fois 1 à r_{1,0}
     load_1 0
     incr
     store_1 0
enddo                           # Après cette boucle r_{1,0} = pn + j
incr                            # Enfin on ajoute 1 à l'accumulateur
write                           # qui contient déjà r_{1,0}
```

4. Soit une machine PRAM avec p registres. On va avoir besoin d'une version modifiée du code ci-dessus pour $j = 1, ..., p$, que l'on notera Q_j, et qui étant donné un entier $r_{1,1}$ utilise uniquement $r_{1,0}$ de sorte que à la fin du calcul $r_{1,0}$ contienne $pr_{1,1} + j + 3$:

```
load# 0
store_1 0                       # On initialise r_{1,0} à0
load_1 1                        # On lit r_{1,1}
do                              # On va ajouter r_{1,1} fois p à r_{1,0}
     load#p                     # On lit p
     do                         # On ajoute p fois 1 à r_{1,0}
          load_1 0
          incr
          store_1 0
     enddo
enddo                           # Après cette boucle r_{1,0} = pn
load#j                          # On lit j
do                              # On ajoute j fois 1 à r_{1,0}
     load_1 0
     incr
     store_1 0
enddo                           # Après cette boucle r_{1,0} = pn + j
incr                            # Enfin on ajoute 3 fois 1 à l'accumulateur
incr
incr
store_1 0
```

On va faire correspondre à la case k du registre j la case $pk + j + 3$ d'une nouvelle machine. La case 2 de la nouvelle machine sert à stocker la valeur de l'accumulateur quand on est face à une instruction

$\texttt{store}_j(k)$ et les case 0 et 1 sert pour le calcul de la somme $pk+j+3$: la case 1 contient n et la case 0 va contenir le $pk+j+3$.
Plus précisément le code de la nouvelle machine est obtenu :
— En remplaçant, pour chaque $j=1,...,r$ et chaque $k\in\mathbb{N}$:
  — $\texttt{load}_j k$ par $\texttt{load}_1 pk+j+3$
  — $\texttt{store}_j k$ par $\texttt{store}_1 pk+j+3$
— En remplaçant, pour chaque $j=1,...,r$ et chaque $k\in\mathbb{N}$:
— $\texttt{load}_j(k)$ par :

```
load#k
store_1 1          # On "passe l'argument k" à Q_j
Q_j                # A la fin r_{1,0} contient la nouvelle adresse
load_1(0)
```

— $\texttt{store}_j(k)$ par :

```
store_1 2          # On stocke le contenu de l'accumulateur
load#k
store_1 1          # On "passe l'argument k" à Q_j
Q_j
load_1 2           # On récupère le contenu de l'accumulateur
store_1(0)         # On stocke le contenu à la nouvelle adresse
```

La machine ainsi modifiée effectue les mêmes calculs, mais avec un seul registre.

5. D'après la question 1 et la définition des fonctions récursives primitives, pour montrer que toutes les fonctions récursives primitives sont PRAM-exécutables, il suffit de montrer la stabilité de l'ensemble des fonctions PRAM-exécutables par composition et par récurrence.

Commençons par la stabilité par composition. Soient $f_1,...,f_n\in\mathcal{F}_p$ PRAM-exécutables et soit $g\in\mathcal{F}_n$ PRAM-exécutable. Soient P_1,... P_n les programmes calculant respectivement à $f_1,...,f_n$ et soit Q un programme calculant g. Soit, pour $i\in\{1,...,n\}$, p_i la taille du registre de f_i et soit q la taille du registre de Q. Notre nouvelle machine PRAM aura $N:=p_1+\cdots+p_n+q+1$ bandes et fonctionne ainsi : elle commence par exécuter chaque P_i en à ceci près que l'on renumérote les bandes de sorte que P_i travaille sur les bandes allant de $p_1+\cdots+p_{i-1}+1$ à $p_1+\cdots+p_i$ et on remplace les `write` par $\texttt{store}_N i$. Ensuite elle exécute Q où on a remplacé les $\texttt{read}_k$ par des $\texttt{load}_N k$ en travaillant sur les bandes de $N-q$ à $N-1$. Cette nouvelle machine exécute bien $g(f_1,...,f_n)$ par construction.

Pour la stabilité par récurrence, soient $g\in\mathcal{F}_p$ et $h\in\mathcal{F}_{p+2}$ toutes deux PRAM-exécutables par des programmes respectifs P et Q à n et m registres. Soit f définie par

$$f(x_1,...,x_p,0)=g(x_1,...,x_p)$$
$$f(x_1,...,x_p,y+1)=h(x_1,...,x_p,y,f(x_1,...,x_p,y))$$

La machine calculant f définie par récurrence à partir de g et h aura $N:=n+m+1$ registres. La case 0 du registre N contiendra à la fin $f(x_1,...,x_n,y)$. La case 1 du même registre contiendra le t qui ira de 0 à $y-1$ afin de calculer $f(x_1,...,x_n,y)$ via une boucle. Enfin la case 2 contiendra le résultat du calcul de Q.
On modifie le programme P comme suit : on remplace les `write` par des $\texttt{store}_N 0$ de sorte qu'à la fin de l'exécution de P, $r_{N,0}$ contienne $f(x_1,...,x_n,0)$.
On modifie le programme Q comme suit : on remplace les $\texttt{read}_{p+1}$ (lecture de y) par $\texttt{load}_N 1$ et les $\texttt{read}_{p+2}$ (lecture de $f(x_1,...,x_p,y)$) par $\texttt{load}_N 0$. Enfin on remplace les `write` par $\texttt{store}_N 2$ (le résultat est donc sto
Le programme calculant f est le suivant :

```
P
read_{p+1}              # Lecture de y
do                      # On "itère" y fois h
     Q
     load_N 2
     store_N 0          # On remplace f(x_1,...,x_p,t) par f(x_1,...,x_p,t+1)
enddo
load_N 0
write
```

On a donc obtenu la stabilité par récurrence et par composition de l'ensemble des fonctions PRAM-exécutables, ce qui comme expliqué plus haut établit que

les fonctions récursives primitives sont PRAM-exécutables.

6. Comme on a qu'un seul registre, on note son contenu $(r_k)_{k\in\mathbb{N}}$. On code le contenu de registre via notre codage Ω des suites finies : on pose $\Omega((r_k)) = \prod_{k\in\mathbb{N}} \pi(k)^r_k$. On code alors la configuration $((x_1, ..., x_p), (r_k)_{k\in\mathbb{N}}, a, s)$ en $\alpha_{p+3}(x_1, ..., x_p, \Omega((r_k)_{k\in\mathbb{N}}), a, s)$. Remarquons que le codage est une bijection de l'ensemble des configurations avec $\mathbb{N}^*$.
7. On montre par induction sur P programme que la fonction qui à n associe le code de la configuration obtenue en effectuant P sur la configuration codée par n est récursive primitive.
   — Le programme vide ne modifie pas la configuration, il correspond donc à la fonction identité qui est primitive récursive.
   — Supposons qu'à un programme P corresponde une fonction récursive primitive f telle que P modifie les codes de configuration selon f, et considérons un programme obtenu en faisant P suivi d'une instruction et voyons la fonction g par laquelle il faut composer f pour obtenir la nouvelle fonction de modification des codes de configurations. Pour simplifier les notations, on pose $b_i = \beta^i_{p+3}(n)$ pour $i = 1, ..., p$, $c = \beta^{p+1}_{p+3}(n)$, $a = \beta^{p+2}_{p+3}(n)$ et $s = \beta^{p+3}_{p+3}(n)$, et on rappelle que la fonction $\delta(x, i)$ qui donne l'exposant de $\pi(i)$ dans x est récursive primitive. Comme on a utilisé un codage multiplicatif, les modifications du registre `store` utilisent la fonction quotient q. Donnons donc les fonctions promises : si P est suivi de
     — `load`$\#k$ on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, c, a, s)$.
     — `load`$_1 k$ on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, c, \delta(c, k), s)$
     — `load`$_1(k)$ on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, c, \delta(c, \delta(c, k)), s)$
     — `store`$_1 k$ on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, q(c \times \pi(k)^a, \pi(k)^{\delta(c,k)}), a, s)$
     — `store`$_1(k)$ on pose $g(n) = \alpha_{p+3}((b_1, ..., b_p, q(c \times \pi(\delta(c, k))^a, \pi(\delta(c, k))^{\delta(c,\delta(c,k))}), a, s)$
     — `incr` on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, c, a + 1, s)$
     — `decr` on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, c, a \dot{-} 1, s)$
     — `read`$_k$ on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, c, b_k, s)$
     — `write` on pose $g(n) = \alpha_{p+3}(b_1, ..., b_p, c, a, a)$

     On vérifie alors que g se comporte comme on veut, de plus à chaque fois g est récursive primitive donc puisque l'ensemble des fonctions récursives primitives est clos par récursion, $g \circ f$ calcule les modifications de configuration du nouveau programme de manière primitive récursive, comme voulu.
   — Supposons qu'à deux programme P et Q corresponde des fonctions récursives primitives respectives f et g telle que P modifie les codes de configuration selon f et Q selon g. Alors le programme

```
P
do
      Q
enddo
```

   modifie la configuration de code n en itérant $\beta^{p+2}_{p+3}(f(n))$ la fonction g, autrement dit la fonction h de modification des configurations est $h(n) = g^{\beta^{p+2}_{p+3}(f(n))}(f(n))$. Or on a vu (exercice 2) que l'itération est récursive primitive donc h est récursive primitive.

   On conclut donc par induction que

les machines PRAM modifient les codes de configuration de manière primitive récursive.

8. Remarquons que la fonction qui à $(x_1, ..., x_p)$ associe le code de la configuration initiale d'une machine PRAM d'entrées $(x_1, ..., x_p)$ est récursive primitive, puisque c'est $\alpha_{p+3}(x_1, ..., x_p, 1, 0, 0)$. Soit donc une fonction f PRAM-exécutable de programme P, soit g la fonction de modification de configuration de P obtenue via la question précédente, alors par construction pour tous $x_1, ..., x_p$ on a

$$f(x_1, ..., x_p) = \beta^{p+3}_{p+3}(g(\alpha_{p+3}(x_1, ..., x_p, 1, 0, 0)))$$

et donc f est récursive primitive. Ainsi toute fonction PRAM-exécutable est récursive primitive, et d'après la question 5 on peut conclure que

toute fonction est PRAM-exécutable ssi elle est récursive primitive.