README Ce fichier n'est qu'un brouillon de réécriture de l'article de Grandjean et Schwentick *Machine-independent characterizations and complete problems for deterministic linear time* pour voir si ces notions peuvent s'étendre au temps $\mathcal{O}\left(n^k\right)$.

C'est plus ou moins une traduction littérale en français et en remplaçant $\mathcal{O}(n)$ par $\mathcal{O}\left(n^k\right)$, pour voir si ça coince à un endroit.

Notations Par abus de notation découlant de la théorie des ensembles, j'écrirai n pour $[\![0, n-1]\!]$ voire pour $[\![1, n]\!]$ quand il n'y aura pas d'ambiguïté. De manière générale, si un terme ressemble à un entier naturel mais qu'il est mis à la place d'un ensemble (typiquement, dans une fonction), il faut le lire comme l'ensemble $[\![0, n-1]\!]$.

Preliminaries (p.198)

RAM data structures (p.198)

Définition 1 (RAM data structures)**.** Soit t un type, c'est-à-dire une signature fonctionnelle ne contenant que des symboles de constantes ou de fonctions unaires.

Une RAM-structure s de type t est un uplet constitué de :

- $n \in \mathbb{N}$ est la taille de la structure ;
- $C \in \mathbb{N}$ pour chaque symbole $C \in t$;
- $f : n \to \mathbb{N}$ pour chaque symbole $f \in t$.

On notera $s.n, s.C, s.f$ les composantes n, C, f de s (cette notation est à rapprocher de l'accès à un attribut ou à une fonction membre en programmation objet).

On dira que s est c-bornée pour $c \in \mathbb{N}$ lorsque $s.C, s.f(i) < cs.n$ pour tous $C, f \in t$ et $i \in n$.

Définition 2 (Fonction de RAM)**.** Soient t_1, t_2 des types.

Une (t_1, t_2)-fonction de RAM Γ est une fonction telle qu'il existe $c_1, c_2 \in \mathbb{N}$, tels que Γ envoie les structures c_1-bornées de type t_1 sur des structures c_2-bornées de type t_2.

On dit que Γ est polynomiale lorsque $\Gamma(s).n = \mathcal{O}\left((s.n)^k\right)$.

Machine RAM (p.200)

La machine RAM reste la même (heureusement). On va utiliser la $\{+\}$-RAM ou des versions un brin plus puissantes comme la $\{+, -, \times, \div k\}$-RAM pour un $k \in \mathbb{N}$ fixé.

Définition 3 (Temps polynomial)**.** On définit $\mathrm{DTIME}_{\mathrm{RAM}}\left(\mathcal{O}\left(n^k\right)\right)$ comme étant l'ensemble des fonctions calculables sur $\{+\}$-RAM en temps $\mathcal{O}\left(n^k\right)$, telles que le nombre de registres utilisés, la longueur des nombres manipulés (y compris les adresses de registres) soient bornés par $\mathcal{O}\left(n^k\right)$.

Réductions affines (p.202)

On laisse inchangées les notions de *transformations affines* (définition 2.3), de *réductions affines* (définition 2.5),de projections affines (définition 2.7). Les théorèmes et lemmes suivants ou intermédiaires sont aussi inchangés. Ils permettront de définir des réductions qui *restent* dans $\mathrm{DTIME}_{\mathrm{RAM}}\left(\mathcal{O}\left(n^k\right)\right)$ pour un k fixé.

Le framework algébrique (p.208)

LSRS (p.208)

Le LSRS en tant que tel n'a pas l'air collé à la définition du temps linéaire. On garde la définition pour le moment.

On doit refaire la définition 3.3.

Pour t un type, on note F_t l'ensemble de symboles de fonctions suivants : $\{1(-), n(-), id(-)\} \cup \{f_C | C \in t\} \cup \{f | f \in t\}$.

Remarque 1. *Soient t un type et S un LSRS pour $f_1, \ldots, f_h$ sur F_t. L'entrée d'un LSRS peut être vue comme étant une RAM-structure s de type t, qu'il* lit *en interprétant les symboles de F_t de la façon suivante :*

- $\forall f \in t_1 : f(i) = \begin{cases} s.f(i) & si\ i < s.n \\ 0 & sinon \end{cases}$
- $\forall C \in t_1 : f_C(i) = s.C$
- $1(i) = 1, n(i) = s.n, id(i) = i$

La sortie du LSRS peut aussi être vue comme une nouvelle structure $s' = S(s)$ de type $\{f_1, \ldots, f_h\}$.

Ici, on a plusieurs possibilités pour adapter le concept de *linéairement représenté* (définition 3.3) à n^k.

Définition 4 (RAM n^k-représentée par LSRS - Proposition 1)**.** Soient t_1, t_2 des types. Soit Γ une (t_1, t_2)-fonction de RAM.

Soit S un LSRS pour $f_1, \ldots, f_h$ sur F_{t_1}.

On dit que Γ est n^k-représentée par S lorsqu'il existe un entier c et une projection affine P tels que, pour chaque structure s c-bornée, S définit des fonctions $f_1, \ldots, f_h : c(s.n)^k \to c(s.n)^k$ telles que $\Gamma(s) = P((s.n)^k, S(s))$ ($S(s)$ est la structure définie par le LSRS).

La modification réside dans le domaine de définition des fonctions et la taille de la sortie dans la projection.

Problèmes 1.

- *Est-ce qu'on capture tout* $DTIME_{RAM}\left(\mathcal{O}\left(n^k\right)\right)$ *?*
- *Est-ce que* $P((s.n)^k, S(s))$ *capture toutes les structures de taille* $\mathcal{O}\left(n^k\right)$ *?*

Si Γ est n^k-représentée par un LSRS alors on dit que Γ est définissable par LSRS.

La définition 3.4 (définition par cas) et les lemmes 3.5 (la définition par cas ne change pas la puissance des LSRS) et 3.6 (composition de fonctions définissables par LSRS reste définissable par LSRS) restent les mêmes.

LRS (p.211)

La définition d'un terme récursif (non numérotée) et d'un LRS (définition 3.7) restent les mêmes. Le lemme 3.8 d'existence d'une solution unique au LRS aussi.

On doit adapter la définition de n^k-représentée par LRS :

Définition 5 (RAM n^k-représentée par LRS - Proposition 1)**.** Soit Γ une fonction de RAM.

Soit E un LSRS $g(x) = \sigma(x)$.

On dit que Γ est n^k-représentée par E lorsqu'il existe un entier c et une projection affine P tels que, pour chaque structure s c-bornée, E définit une fonction $g : c(s.n)^k \to c(s.n)^k$ telle que $\Gamma(s) = P((s.n)^k, E(s)$ ($E(s)$ est la structure définie par le LRS, elle est de type $\{g\}$).

La modification réside dans le domaine de définition de g*.*

Et là, c'est la foire.

On doit vérifier si le théorème principal de l'article tient encore au temps polynomial.

Conjecture 1 (Adaptation du lemme 3.10 (p.212))**.** *Toute fonction de RAM* n^H*-représentée par LSRS est aussi* n^H*-représentable par LRS.*

Démonstration. Ici commence la relecture de la preuve.

Soit Γ une fonction de RAM n^H-représentée par un LSRS S sur F_{t_1} pour $f_0, \ldots, f_{k-1}$, comme dans la définition, et soit P la projection affine associée. Soit s une RAM-structure c-bornée de type t_1. On note $s' = S(s)$ la structure définie par S avec entrée s, et on note $s'' = \Gamma(s) = P\left((s.n)^H, s'\right)$. Pour simplifier les notations, on va simplement écrire n au lieu de $s.n$.

L'idée est de coder les fonctions $f_0, \ldots, f_{k-1} : cn^H \to cn^H$ par une unique fonction g. Pour s'assurer que la fonction *Equal-Predecessor* fonctionne correctement, le codage des f_i doit avoir des domaines disjoints et des images disjointes. Cela peut se faire, pour chaque $i < k$, en n'utilisant que des valeurs congrues à i modulo k pour la fonction f_i. Puisqu'on va en avoir besoin pour coder/décoder les opérations, on va aussi coder la fonction $\div k$ dans g.

On va étendre le domaine de g pour encoder $\Gamma(s).f$ pour chaque symbole de fonction f du type de la structure de sortie, *by function values of a contiguous interval* (? ? ?).

Précisément, g va être définie sur $2kcn^H$ de sorte que :

- pour tout $b \in kcn^H$, on a $g(b) = b \div k$;
- pour tous $a \in cn^H$ et $i \in k$, on a $g\left(kcn^H + ka + i\right) = kcn^H + kf_i(a) + i$ $(\star)$

On a donc besoin d'un moyen de calculer n^H dans la fonction de sortie.
Pour l'instant, pour des raisons de simplicité, puisqu'on sait que la fonction est n^H-représentable, on va supposer qu'on a accès à une fonction $i \mapsto n^H$. Pour ce faire, on peut soit la définir par multiplication en rajoutant H équations (ce qui ne me semble pas naturel, parce qu'on modifie le LSRS à chaque fois qu'on veut changer la taille d'arrivée ?) soit on peut rajouter une fonction constante $i \mapsto n^H$ (on ajoute ou remplace une fonction dans les fonctions de base du LSRS ? Ca veut dire qu'on rajoute un symbole de fonction au type de départ, qui sait déjà ce qu'on va en faire ?)
Problème : apparemment le LSRS sait qu'il représente des fonctions de taille n^H, mais est-ce suffisant pour justifier l'insertion de cette fonction en tant que fonction du domaine ? Apparemment, un même LSRS peut être utilisé pour toutes les tailles de domaines, donc peut-être qu'on peut, au cas par cas, rajouter une fonction en plus selon le domaine d'arrivée ?
(La question ici est de savoir si on peut rajouter naturellement cette fonction constante d'accès direct au max, sans avoir l'air de tricher pour se faciliter la vie.)
Ou bien on peut voir l'ajout de cette fonction comme un ajout de contrainte. C'est l'ajout de cette fonction qui impose que le domaine d'arrivée sera de taille $\mathcal{O}\left(n^k\right)$.

En d'autres termes, pour $b \geqslant kcn^H$ et $b \mod k = j$, on a $g(b) = kcn^H + kf_j\left(\left(b - kcn^H\right) \div k\right) + j$. On va définir la valeur de $g(b)$ pour $b \in 2kcn^H$ par distinction de cas sur $b \mod k$, comme décrit dans (1) à (3) ci-dessous.

1. Renumérotons les symboles de $F_{t_1} = \{f|f \in t_1\} \cup \{f_C|C \in t_1\} \cup \{1(-), n(-), id(-)\}$ ainsi $\{f^0, \ldots, f^{l-1}\}$. On va limiter les occurrences de ces symboles. Premièrement, les symboles $f_0, \ldots, f_{k-1}$ sont remplacés par $f^l, \ldots, f^{l+k-1}$ respectivement. Ensuite, On va introduire l nouvelles fonctions $f_0, \ldots, f_{l-1}$ et l équations pour les définir :

   - Si f^i vient de t_1, alors l'équation associée est $f_i(x) = f^i(x)$;
   - Si $f^i = id$, alors l'équation associée est $f_i(x) = x$;
   - Si $f^i = f_C$, ou 1 ou n, alors l'équation associée est (respectivement) $f_i(x) = C, 1, n$.

   On appellera ces équations des *équations d'entrée* ; elles servent justement à remplacer les entrées du LSRS.

   Enfin, on remplace dans le LSRS S tous les anciens symboles de fonctions (ceux de F_{t_1}) par les nouveaux (les $(f_i)_{i \in l+k}$). Après ces remplacements, S ne contient plus aucune référence à F_{t_1}, sauf pour les équations d'entrée.
2. Les équations de S sont combinées en une seule équation $g(y) = \sigma(y)$ comme suit. Le terme $\sigma(y)$ est principalement une distinction de cas dépendant de y et $y \mod k$.

$$g(y) = \begin{cases} 0 & \text{si } y \leqslant k-1 \\ g(y-k)+1 & \text{si } k \leqslant y < kcn^H \\ \sigma_i(y) & \text{si } y \geqslant kcn^H \text{ et } y \mod k = i \end{cases}$$

   où $\sigma_i(y)$ est un terme récursif qu'on explicitera tout de suite après.

   Notons que les deux premiers cas donnent $g(y) = y \div k$ pour chaque $y < kcn^H$, comme voulu. Cela nous permet d'exprimer $y \mod k$ pour $kcn^H \leqslant y < 2kcn^H$, puisque $(y - kcn^H) - kg(y - kcn^H) = (y - kcn^H) - \sum_{j=1}^{k} g(y - kcn^H)$. (Et alors ? ? ?)

   Ensuite, on a vu que la distinction de cas ne rendait pas le LSRS plus puissant.

   Enfin, on décrit la construction des termes de récurrence $\sigma_i(y)$ (on distingue la variable y du terme de récurrence, de la variable x des équations) :

   - Si E_i est une équation d'entrée, alors $\sigma_i(y)$ est construit comme suit :
     - Si le terme de droite de l'équation est une constante C (éventuellement 1 ou n), alors $\sigma_i(y) = kcn^H + kC + i$;
     - Si le terme de droite de l'équation est x, alors $\sigma_i(y) = kcn^H + kg\left(y - kcn^H\right) + i$;
     - Si le terme de droite de l'équation est $f^i(x)$, alors $\sigma_i(y) = kcn^H + kf^i\left(g\left(y - kcn^H\right)\right) + i$. Justifions pourquoi cette définition de $\sigma_i(y)$ est correcte. On le fait pour le troisième cas ; les deux autres sont plus simples. Soit $b = kcn^H + ka + i$ avec $a < cn^H$. Alors $g(b - kcn) = g(ka + i) = (ka + i) \div k = a$, et $\sigma_i(b) = kcn^H + kf^i(a) + i$, comme voulu.
   - Si E_i est de la forme $f_i(x) = f_j(x) - f_{j'}(x)$, alors $\sigma_i(y)$ est défini par :

$$\sigma_i(y) = \left(g(y-\delta) - kcn^H - j\right) - \left(g(y-\delta') - kcn^H - j'\right) + kcn^H + i$$

     où $\delta = i - j$ et $\delta' = i - j'$ et, par définition d'un LSRS, $i > j, j'$. Pour vérifier que cette expression est correcte, soient $a \in cn^H$, $b = kcn + ka + i$, où $i < k$. Si $g(b - \delta) = g(kcn^H + ka + j) = kcn^H + kf_j(a) + j$ et $g(b - \delta') = g(kcn^H + ka + j') = kcn^H + kf_{j'}(a) + j'$ alors :

$$\left(g\left(b-\delta\right) - kcn^H - j\right) - \left(g\left(b-\delta'\right) - kcn^H - j'\right) = k\left(f_j(a) - f_{j'}(a)\right) + kcn^H + i$$

     Ce qui est ce qu'on voulait.
   - Si E_i est de la forme $f_i(x) = f_j(x) + f_{j'}(x)$, alors son traitement est similaire au cas précédent, à ceci près qu'il faut que l'addition $x + y$ renvoie 0 si $x + y > cn^H$. On redéfinit alors $\sigma_i(y)$:

$$\sigma_i(y) = \begin{cases} \tau(y) + kcn^H + i & \text{si } \tau(y) < kcn^H \\ kcn^H + i & \text{sinon} \end{cases}$$

     où $\tau(y) = \left(g\left(b-\delta\right) - kcn^H - j\right) - \left(g\left(b-\delta'\right) - kcn^H - j'\right)$ avec $\delta = i - j$ et $\delta' = i - j'$.
     La vérification se passe de la même manière que précédemment.

- Si E_i est de la forme $f_i(x) = f_{j'}\left[f_j^{\leftarrow}(x)\right]_x$, où $j < i$ et on suppose sans perte de généralité que $i \leqslant j'$ [1], alors $\sigma_i(y)$ est définie par :

$$\sigma_i(y) = g\left[g^{\leftarrow}(y-\delta)+\delta'\right]_y - j' + i$$

où $\delta = i - j$ et $\delta' = j' - j$.

Pour justifier ce remplacement, on doit s'assurer que le codage de plusieurs fonctions en une seule ne cause par d'effets de bord quand on utilise *Equal-Predecesor*. Ce qui est crucial ici, c'est que, pour chaque $i < k$, les valeurs de g qui codent f_i soient congruentes à i modulo k. Pour être plus précis, soit $b = kcn^H + ka + i$, avec $a < cn^H$. Alors $b - \delta = kcn^H + ka + i - (i-j) \underset{i>j}{=} kcn^H + ka + j$. On a deux sous-cas à étudier :

  - $f_j^{\leftarrow}(a) = a$. Dans ce cas, pour aucun $a' < a$, on n'a $f_j(a') = f_j(a)$, donc il n'y a pas de $a' < a$ pour lequel $g\left(kcn^H + ka' + j\right) = g\left(kcn^H + ka + j\right)$. La définition de g assure que, pour chaque $e \geqslant kcn^H$, on a $(g(e) \mod k = j \Leftrightarrow e \mod k = j)$ et $\forall e, e' \in 2kcn^H$, si $g(e) = g(e')$, alors soit $e, e' < kcn^H$, soit $e, e' \geqslant kcn^H$. Donc $g^{\leftarrow}\left(kcn^H + ka + j\right) = kcn^H + ka + j$ et $g^{\leftarrow}(b-\delta) + \delta' = kcn^H + ka + j' \geqslant kcn^H + ka + i = b$. Ainsi :

$$\begin{aligned}
\sigma_i(b) &= g\left[g^{\leftarrow}(b-\delta)+\delta'\right]_b - j' + i && (1)\\
&= \left(kcn^H + ka + j'\right) - j' + i && (2)\\
&= kcn^H + ka + i && (3)\\
&= kcn^H + kf_{j'}\left[f_j^{\leftarrow}(a)\right]_a + i && (4)\\
&= kcn^H + kf_i(a) + i && (5)\\
&= g(b), && (6)
\end{aligned}$$

  comme voulu.

  - $f_j^{\leftarrow}(a) = a'$ pour un certain $a' < a$. Dans ce cas, $g\left(kcn^H + ka + j\right) = kcn^H + ka' + j$. En conséquence :

$$g^{\leftarrow}(b-\delta) + \delta' = kcn^H + ka' + j' < kcn^H + ka + i = b$$

  Donc :

$$\begin{aligned}
\sigma_i(b) &= g\left[g^{\leftarrow}(b-\delta)+\delta'\right]_b - j' + i && (1)\\
&= g\left(kcn^H + ka' + j'\right) - j' + i && (2)\\
&= \left(kcn^H + kf_{j'}(a') + j'\right) - j' + i && (3)\\
&= kcn^H + kf_{j'}(a') + j' + i && (4)\\
&= kcn^H + kf_{j'}\left[f_j^{\leftarrow}(a)\right]_a + i && (5)\\
&= kcn^H + kf_i(a) + i && (6)\\
&= g(b), && (7)
\end{aligned}$$

  comme souhaité.

3. Maintenant, on complète le LRS pour g. Pour une question de simplicité, on va supposer que t_2 ne contient que le symbole de constante n et un seul symbole de fonction h. Soient $j < k$ et α une fonction affine tels que, pour toute structure s, on ait $\Gamma(s).n = P\left((s.n)^H, s'\right).n = s'.f_j\left(\alpha\left((s.n)^H\right)\right)$, et soient $i < k$ et A une fonction affine tels que, pour toute structure s et tout $a < \Gamma(s).n$, on ait $\Gamma(s).h(a) = P\left((s.n)^H, s'\right).h(a) = s'.f_i\left(A\left((s.n)^H, a\right)\right)$ [2].

   On a construit g de telle manière que toutes les valeurs de fonctions $f_i(a)$ sont, en quelque sorte, disponibles dans g, mais on a encore deux problèmes à résoudre. Premièrement, les $f_i(a)$ apparaissent uniquement sous

1. Si $i > j'$ alors on doit rajouter une nouvelle fonction f_l à S, telle que $l > i$, et définie par la nouvelle équation $f_l(x) = f_{j'}$ et remplacer E_i par $f_i(x) = f_l\left[f_j^{\leftarrow}(x)\right]_x$

2. Ça veut juste dire : on donne des noms aux éléments qui permettent de définir la structure $\Gamma(s)$; ces fonctions affines α, A et ces entiers i, j existent par définition d'une fonction n^H-représentable par un LSRS. Reste à voir si cette définition a bien du sens, mais si ça a du sens, alors il n'y a pas de problème ici.

une forme codée ; deuxièmement, elles ne forment pas un intervalle contigu mais sont éparpillées modulo k. Il faut donc, avant d'extraire les valeurs à l'aide d'une projection affine bien choisie, on doit décoder les valeurs des fonctions et les ramener dans un même intervalle. Pour ça, on élargit le domaine de g à $(2k+2)cn^H$ et on complète la définition de g :

$$g(y) = \begin{cases} \text{comme avant} & \text{si } y < 2kcn^H \\ g\left[g\left[k\left(y - 2kcn^H\right) + kcn^H + i\right]_y - kcn^H\right]_y & \text{si } 2kcn^H \leqslant y < (2k+1)cn^H \\ g\left[g\left[k\left(y - (2k+1)cn^H\right) + kcn^H + j\right]_y - kcn^H\right]_y & \text{si } (2k+1)cn^H \leqslant y < (2k+2)cn^H \end{cases}$$

Il découle de l'équation $(\star)$ (la définition de g sur kcn^H) et de cette définition que, pour tout $a < cn^H$, on a[3] :

$$\begin{aligned} g\left(2kcn^H + a\right) &= g\left[g\left[k\left(\left(2kcn^H + a\right) - 2kcn^H\right) + kcn^H + i\right]_{2kcn^H+a} - kcn^H\right]_{2kcn^H+a} && (1) \\ &= g\left[g\left[ka + kcn^H + i\right]_{2kcn^H+a} - kcn^H\right]_{2kcn^H+a} && (2) \\ &= g\left[g\left(ka + kcn^H + i\right) - kcn^H\right]_{2kcn^H+a} && (3) \\ &= g\left[\left(kcn^H + kf_i(a) + i\right) - kcn^H\right]_{2kcn^H+a} && (4) \\ &= g\left[kf_i(a) + i\right]_{2kcn^H+a} && (5) \\ &= f_i(a) && (6) \end{aligned}$$

De la même manière, on obtient, pour $a < cn^H$, $g\left((2k+1)cn^H + a\right) = f_j(a)$.

Maintenant, il est aisé de définir une projection affine P' qui extrait $\Gamma(s)$ à partir de s''' de type $\{g\}$, définie par le LRS :

$$\Gamma(s).n = s'''.g\left(\alpha\left(cn^H\right) + (2k+1)cn^H\right)$$ [4]

et[5] [6] :

$$\begin{aligned} \Gamma(s).h(a) &= P\left(n^H, s'\right).h(a) && (1) \\ &= s'.f_i\left(A\left(n^H, a\right)\right) && (2) \\ &= s'''.g\left(2kcn^H + A\left(n^H, a\right)\right) && (3) \end{aligned}$$ [7]

□

Bilan La démonstration a l'air de marcher ! !

3. Analysons :

- (1) → (2) : parce que soustraction propre.
- (2) → (3) : par définition de l'opérateur *application bornée*.
- (3) → (4) : parce que $ka + kcn^H + i < 2kcn^H$ donc on reprend la définition de la fonction g sur l'intervalle $2kcn^H$.
- (4) → (5) : *because* soustraction propre.
- (5) → (6) : parce que $f_i(a) < cn^H$ et on a le bon décalage avec i.

4. C'est bien ce qu'il nous faut, parce que $\alpha' : a \mapsto \alpha(ca) + (2k+1)ca$ est bien une fonction affine telle que $P'\left(n^H, s'\right).n = s'''.g\left(\alpha'\left(n^H\right)\right)$.
5. On rappelle que s' est la structure résultante du LSRS originel, de type $\{f_0, \ldots, f_{k-1}\}$, qu'on a compactée dans une seule structure de type $\{g\}$.
6. Analysons :

- (1) → (2) : par définition de A et i.
- (2) → (3) : parce qu'on a $f_i(a) = g\left(2kcn^H + a\right)$ par construction de g.

7. C'est bien ce qu'il nous faut parce que $A' : a, b \mapsto 2kcb + A(b, a)$ est bien une fonction affine telle que $P'\left(n^H, s'\right).h(a) = s'''.g(A'(n^H, a))$.

LRS et temps linéaire (p.216)

Conjecture 2. *Soit* Γ *une fonction de RAM. Les propositions suivantes sont équivalentes :*

1. $\Gamma \in DTIME_{RAM}\left(\mathcal{O}\left(n^H\right)\right)$.
2. Γ *est* n^H*-représentée par un LSRS.*
3. Γ *est* n^H*-représentée par un LRS.*

On a montré (2) $\Rightarrow$ (3) juste avant.

(3) $\Rightarrow$ (1)

Lemme 1. *Si une fonction de RAM* Γ *est* n^H*-représentée par un LRS* E *alors* $\Gamma \in DTIME_{RAM}\left(\mathcal{O}\left(n^H\right)\right)$.

Démonstration. Soit Γ une fonction de RAM n^H-représentée par une équation E, et soit P la projection affine correspondante. Pour des raisons de simplicité, on va considérer que le type d'entrée de Γ ne contient qu'un seul symbole de fonction f_{in}. Rappelons-nous que E est une équation de la forme $g(x) = \sigma(x)$, où $\sigma(x)$ est une terme de récursion, et qu'il existe une constante c (et H ?) telles que la sortie $s' = \Gamma(s)$ peut être extraite de $(s.n)^H$ [8]et $g : cn^H \to cn^H$.

Au lieu de définir formellement la RAM pour Γ, on va donner un algorithme qui pourra facilement être converti en une RAM à plusieurs mémoires, puis en une RAM à une mémoire [9].

Sa structure de données associée sera constituée de variables p, x et, en plus de la structure d'entrée s et de la structure de sortie s', de quatre tableaux à une dimension $F_{\text{in}}, G, G_{\text{inverse}}$ et EP.

Dans p, on va stocker $c(s.n)^H$ [10]qui borne le domaine de g, définie par E et s. Ensuite, on a décrire le sens de $F_{\text{in}}, G, G_{\text{inverse}}$ et EP. Pour des indices plus grands que $s.n$, F_{in} vaut 0 ; pour les indices plus petits que $s.n$, F_{in} contient la valeur de $s.f$. Le calcul principal se fait en p étapes, numérotés de 0 à $p - 1 = cn^H - 1$.

Après le tour n°i, on devrait avoir les invariants suivants :

(a) pour tout $j < p$, $G[j] = \begin{cases} g(j) & \text{si } j \leqslant i \\ j & \text{si } i < j < p \end{cases}$;

(b) $EP[j] = g^{\leftarrow}(j)$ pour chaque $j \leqslant i$;

(c) pour tout $j < p$, $G_{\text{inverse}}[j] = \begin{cases} \max\left(\{l \leqslant i | g(l) = j\}\right) & \text{si un tel } l \text{ existe} \\ p & \text{sinon} \end{cases}$.

Le calcul des valeurs de $G[i]$ est assez immédiat. On associe à chaque terme de récurrence $\sigma(x)$ un terme de programmation $\sigma[x]$ de la façon suivante :

- Si $\sigma(x)$ est $1, n, x$ alors $\sigma[x]$ est $1, n, x$, respectivement ;
- Si $\sigma(x) = g^{\leftarrow}(x - \delta)$ pour une certain δ, alors $\sigma[x] = EP[x - \delta]$;
- Si $\sigma(x) = g[\tau(x)]_x$ pour un certain terme de récurrence $\tau(x)$, alors $\sigma[x] = G[\tau[x]]$;
- Si $\sigma(x) = f_{\text{in}}(\tau(x))$ pour un certain terme de récurrence $\tau(x)$, alors $\sigma[x] = F_{\text{in}}[\tau[x]]$;
- Si $\sigma(x) = \tau_1(x) * \tau_2(x)$ pour des termes de récurrence $\tau_1(x)$ et $\tau_2(x)$, alors $\sigma[x] = \tau_1[x] * \tau_2[x]$.

Au tour i, on calcule $G[i]$ en évaluant $\sigma[i]$. La seule difficulté est de bien évaluer les termes $g^{\leftarrow}(x - \delta)$, qui devrait prendre plus qu'un nombre constant d'étapes pour être évalué, suivant la méthode directe. Au lieu de faire ça, on va utiliser le tableau EP qui contient, après le tour i, la valeur de $g^{\leftarrow}(j)$ pour chaque $j \leqslant i$. G_{inverse} contient toujours une inverse partielle de g sur $[\![0, j]\!]$ et est utilisé pour calculer EP.

8. Donc on a clairement besoin d'une fonction $i \mapsto n^H$. Sauf qu'il faut en limiter l'accès pour éviter de s'en servir de manière à obtenir des nombres trop gros ? Ou ce n'est pas notre problème parce que de toute façon, si le calcul dépasse les bornes imposées, on n'est plus dans la même classe de complexité ?

9. Est-ce toujours vrai pour une RAM polynomiale ?

10. Du coup on n'a plus besoin de la fonction $i \mapsto n^H$? Dans l'article, on a déjà accès à $cs.n$ pour simuler la machine. D'où vient cette connaissance ?

Voici l'algorithme pour Γ :

```
Input: s
// Initializations
p := c(s.n)^H ;
EP[0] := 0 ;
for j = 0 to s.n − 1 do
  | F_in[j] := s.f(j);
end
for j = s.n to p − 1 do
  | F_in[j] := 0;
end
for j = 0 to p − 1 do
  | G[j] := j;
  | G_inverse[j] := p ;
end
// Main loop
for i = 0 to p − 1 do
  | G[i] = σ[i] ;
  | EP[i] := min ({i, G_inverse[G[i]]}) ;
  | G_inverse[G[i]] := i ;
end
Output: Compute s'' by applying the affine projection P_{(s.n)^H} to the structure s' = (p, G)
```

On montre par induction que les invariants (a), (b), (c) sont maintenus pendant chaque étape de calcul. Bien évidemment, les invariants sont respectés après l'initialisation, c'est-à-dire, avant le tour 0. Pour l'induction, supposons que les invariants (a), (b), (c) sont respectés *avant* le tour $i \in p$. On montre qu'ils sont valides *après* le tour i, et donc avant le tour $i + 1$.

- On a $G[i] = g(i)$ parce que, par induction, $G[j] = g[j]_i$ pour tout $j \in p$, et $EP[j] = g^{\leftarrow}(j)$ pour tout $j \in i$; ainsi, l'opérateur d'applicaiton bornée et la fonction *Equal-Predecesor* sont évaluées correctement.
- L'assignation $EP[i] := \min(\{i, G_{\text{inverse}}[G[i]]\})$ implique que $EP[i] = g^{\leftarrow}(i)$ par induction.
- Il est immédiat de voir que (c) est maintenu par l'assignation $G_{\text{inverse}}[G[i]] := i$.

Donc (a), (b) et (c), et en particulier (a), sont maintenus, donc le programme est correct.

Le nombre d'étapes pour évaluer un terme de récurrence est linéaire en la longueur du terme. Comme le terme de récurrence de E est fixé, il s'agit d'un nombre constant d'étapes. Ainsi, $G[i]$ n'a besoin que d'un nombre constant d'étapes, de sorte que le programme tourne en temps $\mathcal{O}(p) = \mathcal{O}\left(n^H\right)$.

$\geqslant$ => C'est le bien.

$\geq$ => C'est pour les flemmards.

□