M2 LFMI
BROUILLON

BEURIER Erwan

Année 2015-2016

Table des matières

Chapitre 1

LSRS et temps polynomial

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.

Je n'aime pas que les liens hypertextes soient trop visibles, mais je n'ai pas encore cherché comment on pouvait les rendre discrets, donc pour l'instant, ils sont simplement invisibles. Ne pas hésiter à passer la souris sur des mots comme *ici*, *plus bas*, etc ; ils seront très probablement munis d'un lien hypertexte. Les références à des sections, définitions, théorèmes, etc. sont aussi cliquables.

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 le domaine de départ ou d'arrivée d'une fonction), il faut le lire comme l'ensemble $[0, n-1]$.

1.1 Preliminaries

(p.198)

1.1.1 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 :

- *$n \in \mathbb{N}$ qui 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 [1]*.*

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

1.1.2 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 [2].

Définition 3 (Temps polynomial)**.** *On définit $DTIME_{RAM}\left(n^H\right)$ comme étant l'ensemble des fonctions calculables sur $\{+\}$-RAM en temps $\mathcal{O}\left(n^K\right)$, telles que le nombre de registres utilisés, les valeurs entières manipulées (y compris les adresses de registres) soient bornés par $\mathcal{O}\left(n^K\right)$.*

1.1.3 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 $\mathrm{DTIME}_{\mathrm{RAM}}\left(n^H\right)$ pour un k fixé.

Définition 4 (Fonction affine non-décroissante)**.** *On appellera* fonction affine non-décroissante*, ou plus simplement,* fonction affine*, une fonction A de la forme $A(x_1, \ldots, x_k) = a_0 + a_1 x_1 + \cdots + a_k x_k$, telle que soit $a_1, \ldots, a_k \geqslant 0$ et $a_0 \in \mathbb{Z}$, soit $a_1, \ldots, a_k = 0$ et $a_0 \geqslant 0$.*

Définition 5 (Transformation affine)**.** *Soit T une fonction de RAM qui envoie des RAM-structures de type t_1 sur des RAM structures de type t_2.*

On dit que T est une transformation affine lorsque :

REMPLIR ICI!!!!!!!!!!!!

-

1.2 Le framework algébrique

(p.208)

1. On rappelle que "c-borné" ne concerne que la structure par rapport à sa propre taille ; ici on ne compare pas la taille de l'entrée et de la sortie
2. L'ajout de ces opérations ne rend pas la $\{+\}$-RAM plus puissante [2], [1].

1.2.1 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.

Définition 6 (Application bornée et *equal-predecesor*)**.** *Pour $f : n \to \mathbb{N}$, on définit deux opérations :*

- *L'application bornée :*
$$f[x]_y = \begin{cases} f(x) & si\ x < y \\ x & sinon \end{cases}$$
- *L'opération* equal-predecesor *:*
$$f^{\leftarrow} = \begin{cases} \max\left(\{y < x | f(x) = f(y)\}\right) & si\ un\ tel\ y\ existe \\ x & sinon \end{cases}$$

Pour $g, g' : n \to \mathbb{N}$, on combine ces deux opérations pour en créer une troisième, opération de récursion *:*

$$f(x) = g'\left[g^{\leftarrow}(x)\right]_x$$

$f(x) = g'(y)$ où y est de plus grand z tel que $g(x) = g(z)$, ou $f(x) = x$ si un tel y n'existe pas.

Définition 7 (LSRS)**.** *Soit F un ensemble de symboles de fonctions (dites* fonctions de base*), soient $f_1, \ldots, f_k$ des symboles de fonctions qui n'apparaissent pas dans F. Pour $i \leqslant k$, notons $F_i = F \cup \{f_1, \ldots, f_i\}$.*

*Un LSRS (*Linear Simultaneous Recursion Scheme*) S sur $f_1, \ldots, f_k$ et F est une suite de k équations $(E_i)_{i \in k}$ dont chacune est de l'une des deux formes suivantes :*

- *(opération) $f_i(x) = g(x) * g'(x)$ où $g, g' \in F_{i-1}$ et $* \in \{+, -, \times\}$* [3]
- *(récursion) $f_i(x) = g'\left[g^{\leftarrow}(x)\right]_x$ où $g' \in F_k$ et $g \in F_{i-1}$*

On doit refaire la définition 3.3 de l'article (la définition de *linéairement représentable*).
Pour t un type, on note F_t l'ensemble des 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 $f_1, \ldots, f_k$ 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 :

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

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

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

Définition 8 (RAM n^K-représentée par LSRS - Proposition 1 [4])**.** *Soient t_1, t_2 des types. Soit Γ une (t_1, t_2)-fonction de RAM.*

Soit S un LSRS pour $f_1, \ldots, f_k$ 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 $f_1, \ldots, f_k : c(s.n)^K \to c(s.n)^K$ 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.

Problème 1. Est-ce qu'on capture tout $\mathrm{DTIME}_{\mathrm{RAM}}\left(n^H\right)$?

Problème 2. 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.

3. Les opérations peuvent varier, il est dit dans [1] que l'on peut choisir n'importe quelle opération binaire calculable en temps linéaire sur machine de Turing, voire la multiplication. Nous verrons d'ailleurs par la suite que, pour des raisons de facilité, nous aurons besoin de la multiplication.

4. Il pourra à l'avenir y avoir d'autres possibilités mais cette définition semble bien élargir la définition pour le temps linéaire.

1.2.2 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 9 (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^K*-représentée par LSRS est aussi* n^K*-représentable par LRS.*

Démonstration. Ici commence la relecture de la preuve.

Soit Γ une fonction de RAM n^K-représentée par un LSRS S sur F_{t_1} pour $f_0, \dots, f_{k-1}$, 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 $s'' = \Gamma(s) = P\left((s.n)^K, s'\right)$. Pour simplifier les notations, on va simplement écrire n au lieu de $s.n$.

L'idée est de coder les fonctions $f_0, \dots, f_{k-1} : cn^K \to cn^K$ 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 $2kcn^K$ de sorte que :

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

On a donc besoin d'un moyen de calculer n^K dans la fonction de sortie.
Pour l'instant, pour des raisons de simplicité, puisqu'on sait que la fonction est n^K-représentable, on va supposer qu'on a accès à une fonction $i \mapsto n^K$. 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 $i \mapsto n^K$ *(on ajoute ou remplace une fonction dans les fonctions de base du LSRS ? Ça 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^K, 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 $\mathcal{O}\left(n^K\right)$.
Solution Dans la démonstration originale, kcn est calculé à la volée dans le LRS. Il s'agit d'une abréviation : $cn = \underbrace{n(y) + \cdots + n(y)}_{c \text{ fois}}$, et $kcn = \underbrace{cn + \cdots + cn}_{k \text{ fois}}$.
La constante c est toujours une valeur explicite, donnée par définition de la fonction. On peut donc construire ce terme. Idem pour k, qui est le nombre d'équations.
Dans notre cas, on ne peut pas faire un terme similaire pour n^K, car on ne peut pas autoriser un terme dont la longueur dépend de l'entrée (un LRS est défini par une équation fixée dès le départ). Il faut donc qu'on ajoute la multiplication comme opération de base de la définition par LRS, et on pourra écrire un terme similaire : $n^K = \underbrace{n(y) \times \cdots \times n(y)}_{K \text{ fois}}$, puis $cn^K = \underbrace{n^K + \cdots + n^K}_{c \text{ fois}}$, et $kcn^K = \underbrace{cn^K + \cdots + cn^K}_{k \text{ fois}}$.
L'ajout de la multiplication ne rend pas le LSRS, le LRS et la RAM plus puissants, d'après la partie 3.1 et la proposition 2.1 de [1].

En d'autres termes, pour $b \geqslant kcn^K$ et $b \mod k = j$, on a $g(b) = kcn^K + kf_j\left((b - kcn^K) \div k\right) + j$. On va définir la valeur de $g(b)$ pour $b \in 2kcn^K$ 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 $y \mod k$.

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

   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 $y < kcn^K$, comme voulu. Cela nous permet d'exprimer $y \mod k$ pour $kcn^K \leqslant y < 2kcn^K$, puisque $(y - kcn^K) - kg(y - kcn^K) = (y - kcn^K) - \sum_{j=1}^{k} g(y - kcn^K)$. (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 $\sigma_i(y) = kcn^K + kC + i$;
     - Si le terme de droite de l'équation est x, alors $\sigma_i(y) = kcn^K + kg\left(y - kcn^K\right) + i$;
     - Si le terme de droite de l'équation est $f^i(x)$, alors $\sigma_i(y) = kcn^K + kf^i\left(g\left(y - kcn^K\right)\right) + i$.

     Justifions pourquoi cette définition de $\sigma_i(y)$ est correcte. On le fait pour le troisième cas ; les deux autres sont plus simples. Soit $b = kcn^K + ka + i$ avec $a < cn^K$. Alors $g(b - kcn) = g(ka + i) = (ka + i) \div k = a$, et $\sigma_i(b) = kcn^K + kf^i(a) + i$, 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 :

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

     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 $a \in cn^K$, $b = kcn + ka + i$, où $i < k$. Si $g(b - \delta) = g(kcn^K + ka + j) = kcn^K + kf_j(a) + j$ et $g(b - \delta') = g(kcn^K + ka + j') = kcn^K + kf_{j'}(a) + j'$ alors :

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

     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 $x + y > cn^K$. On redéfinit alors $\sigma_i(y)$:

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

     où $\tau(y) = \left(g\left(b - \delta\right) - kcn^K - j\right) - \left(g\left(b - \delta'\right) - kcn^K - j'\right)$ 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 $i \leqslant j'$ [5] , 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 $b = kcn^K + ka + i$, avec $a < cn^K$. Alors $b - \delta = kcn^K + ka + i - (i-j) \underset{i>j}{=} kcn^K + ka + j$. 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 $g\left(kcn^K + ka' + j\right) = g\left(kcn^K + ka + j\right)$. La définition de g assure que, pour chaque $e \geqslant kcn^K$, on a $(g(e) \mod k = j \Leftrightarrow e \mod k = j)$ et $\forall e, e' \in 2kcn^K$, si $g(e) = g(e')$, alors soit $e, e' < kcn^K$, soit $e, e' \geqslant kcn^K$. Donc $g^{\leftarrow}\left(kcn^K + ka + j\right) = kcn^K + ka + j$ et $g^{\leftarrow}(b-\delta) + \delta' = kcn^K + ka + j' \geqslant kcn^K + ka + i = b$. Ainsi :

$$\begin{aligned}
\sigma_i(b) &= g\left[g^{\leftarrow}(b-\delta) + \delta'\right]_b - j' + i && (1.1)\\
&= \left(kcn^K + ka + j'\right) - j' + i && (1.2)\\
&= kcn^K + ka + i && (1.3)\\
&= kcn^K + kf_{j'}\left[f_j^{\leftarrow}(a)\right]_a + i && (1.4)\\
&= kcn^K + kf_i(a) + i && (1.5)\\
&= g(b), && (1.6)
\end{aligned}$$

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

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

    Donc :

$$\begin{aligned}
\sigma_i(b) &= g\left[g^{\leftarrow}(b-\delta) + \delta'\right]_b - j' + i && (1.1)\\
&= g\left(kcn^K + ka' + j'\right) - j' + i && (1.2)\\
&= \left(kcn^K + kf_{j'}(a') + j'\right) - j' + i && (1.3)\\
&= kcn^K + kf_{j'}(a') + j' + i && (1.4)\\
&= kcn^K + kf_{j'}\left[f_j^{\leftarrow}(a)\right]_a + i && (1.5)\\
&= kcn^K + kf_i(a) + i && (1.6)\\
&= g(b), && (1.7)
\end{aligned}$$

    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 $\Gamma(s).n = P\left((s.n)^K, s'\right).n = s'.f_j\left(\alpha\left((s.n)^K\right)\right)$, et soient $i < k$ et A une fonction affine tels que, pour toute structure s et tout $a < \Gamma(s).n$, on ait $\Gamma(s).h(a) = P\left((s.n)^K, s'\right).h(a) = s'.f_i\left(A\left((s.n)^K, a\right)\right)$ [6].

   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

5. 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 $f_i(x) = f_l\left[f_j^{\leftarrow}(x)\right]_x$

6. 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^K-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, décoder les valeurs des fonctions et les ramener dans un même intervalle. Pour ça, on élargit le domaine de g à $(2k+2)cn^K$ et on complète la définition de g :

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

Il découle de l'équation $(\star)$ (la définition de g sur kcn^K) et de cette définition que, pour tout $a < cn^K$, on a [7] :

$$\begin{aligned} g\left(2kcn^K + a\right) &= g\left[g\left[k\left(\left(2kcn^K + a\right) - 2kcn^K\right) + kcn^K + i\right]_{2kcn^K+a} - kcn^K\right]_{2kcn^K+a} && (1.1)\\ &= g\left[g\left[ka + kcn^K + i\right]_{2kcn^K+a} - kcn^K\right]_{2kcn^K+a} && (1.2)\\ &= g\left[g\left(ka + kcn^K + i\right) - kcn^K\right]_{2kcn^K+a} && (1.3)\\ &= g\left[\left(kcn^K + kf_i(a) + i\right) - kcn^K\right]_{2kcn^K+a} && (1.4)\\ &= g\left[kf_i(a) + i\right]_{2kcn^K+a} && (1.5)\\ &= f_i(a) && (1.6) \end{aligned}$$

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

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 :

$$\Gamma(s).n = s'''.g\left(\alpha\left(cn^K\right) + (2k+1)cn^K\right)$$ [8]

et [9] [10] :

$$\begin{aligned} \Gamma(s).h(a) &= P\left(n^K, s'\right).h(a) && (1.1)\\ &= s'.f_i\left(A\left(n^K, a\right)\right) && (1.2)\\ &= s'''.g\left(2kcn^K + A\left(n^K, a\right)\right) \text{ [11]} && (1.3) \end{aligned}$$

□

1.2.3 Bilan

La démonstration a l'air de marcher ! !

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

8. 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 $P'\left(n^K, s'\right).n = s'''.g\left(\alpha'\left(n^K\right)\right)$.

9. 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 $\{g\}$.

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

11. 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 $P'\left(n^K, s'\right).h(a) = s'''.g(A'(n^K, a))$.

1.3 LRS et temps linéaire

(p.216)

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

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

On a montré (2) ⇒ (3) juste avant. La preuve de (3) ⇒ (1) se trouve ci-dessous. La preuve de (1) ⇒ (2) se trouve plus loin.

1.3.1 (3) ⇒ (1)

Conjecture 3. *Si une fonction de RAM Γ est n^K-représentée par un LRS E alors $\Gamma \in DTIME_{RAM}\left(n^K\right)$.*

Démonstration. Soit Γ une fonction de RAM n^K-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 $(s.n)^K$ [12]et $g : cn^K \to cn^K$.

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 [13].

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 $c(s.n)^K$ [14]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 $p - 1 = cn^K - 1$.

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.

12. Donc on a clairement besoin d'une fonction $i \mapsto n^K$. 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 ?
13. Est-ce toujours vrai pour une RAM polynomiale ?
14. Du coup on n'a plus besoin de la fonction $i \mapsto n^K$? 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
$p := c(s.n)^K$;
$EP[0] := 0$;
for $j = 0$ *to* $s.n - 1$ **do**
  $F_{\text{in}}[j] := s.f(j)$;
end
for $j = s.n$ *to* $p - 1$ **do**
  $F_{\text{in}}[j] := 0$;
end
for $j = 0$ *to* $p - 1$ **do**
  $G[j] := j$;
  $G_{\text{inverse}}[j] := p$;
end
// Main loop
for $i = 0$ *to* $p - 1$ **do**
  $G[i] = \sigma[i]$;
  $EP[i] := \min(\{i, G_{\text{inverse}}[G[i]]\})$;
  $G_{\text{inverse}}[G[i]] := i$;
end
Output: Compute s'' by applying the affine projection $P_{(s.n)^K}$ 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'application 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 $\mathcal{O}(p) = \mathcal{O}\left(n^K\right)$.

□

1.3.2 Calculable en temps polynomial ⇒ LSRS

CONJECTURE 4. *Si une fonction de RAM Γ est calculable en temps $\mathcal{O}\left(n^K\right)$ sur une $\{+,-\}$ [15]-RAM M, alors Γ est n^K-représentable par un LSRS.*

Démonstration. Soit Γ une fonction de RAM. Pour des raisons de simplicité, on va supposer que les types d'entrée et de sortie de la fonction n'ont qu'un seul symbole de fonction f. Soit M une RAM calculant Γ comme stipulé dans les hypothèses. Soit c tel que le temps de calcul est borné par cn^K sur des entrées $c(\Gamma)$-bornées [16]de taille n.

On va construire un LSRS qui utilise des fonctions I, A, B, R_A, N, qui décrivent l'état courant de la RAM *avant* chaque étape x, de la façon suivante :

- $I(x)$ contient le numéro courant d'instruction du programme de la RAM ;
- $A(x), B(x), N(x)$ contiennent les valeurs des registres A, B, N de M ;
- $R_A(x)$ contient la valeur du registre dont l'adresse est actuellement dans le registre A.

Par commodité, on va aussi utiliser des fonctions I', A', B', R'_A, N', qui décrivent l'état de la RAM *après* l'étape x.

En définissant $I(x)$ par distinction de cas, une bonne partie de la simulation de la RAM est immédiate. Par exemple, si M exécute $A := A + B$ alors les équations doivent forcer $A'(x) = A(x) + B(x)$. Le principal problème réside dans l'instruction $A := R_A$, qui récupère le contenu d'un registre. Notons que les fonctions définies dans S n'encodent pas explicitement les valeurs de tous les registres de M à chaque étape t mais seulement le contenu du registre dont l'indice est contenu dans A. Pour obtenir la valeur de $R_A(t)$, on doit trouver le dernier instant avant t auquel le registre A avait la valeur $A(t)$. Si un tel instant n'existe pas, on doit se référer à l'entrée. Rappelons que $s.f(i)$ est stocké dans R_i au début du calcul. C'est pour cela que l'opération de récursion entre en jeu [17].

Pour faciliter les instructions $A := R_A$, on va diviser le domaine en trois sous-domaines : $[\![0, cn^K - 1]\!]$, $[\![cn^K, 2cn^K - 1]\!]$ et $[\![2cn^K, 3cn^K - 1]\!]$. On va utiliser le premier intervalle pour stocker $s.f$, le deuxième pour simuler le calcul de M, et le troisième pour extraire la fonction de sortie ; pour ce faire, on va utiliser R_A de manière à encoder la sortie (c'est un usage différent de celui introduit).

On va s'autoriser la définition par cas (on a vu dans un lemme précédent que ça ne rendait pas le LSRS plus puissant) et quelques opérations simples qui ne sont pas directement disponibles dans la définition d'un LSRS. Le système qu'on va proposer n'est pas un LSRS à proprement parler, mais sa traduction en véritable LSRS est immédiate, quoiqu'elle nécessite quelques fonctions en plus.

Pour simplifier la présentation, on va présenter les définitions des fonctions sur les trois intervalles ; elles peuvent être combinées via un LSRS.

Dans un LSRS, l'ordre est important. Ici, les fonctions doivent être présentées dans cet ordre : I, A, B, N, R_A, I', A', B', N', R'_A.

Dans l'article d'origine [1] de ce brouillon, les auteurs prennent la convention que l'addition est bornée ; si $a + b > cn$ où cn est la taille du domaine, alors $a + b$ est en fait évalué à 0 dans le LSRS. On peut utiliser cette convention pour simuler une définition par cas sur la taille entière du domaine. Si le LSRS est défini sur $3cn^K$ alors le test $x < cn^K$ peut se simuler avec le test : $x = 0$ ou $x + x + x > 0$.

Ainsi, avant de définir les autres fonctions, intéressons-nous à un moyen d'obtenir cn^K dans le LSRS, ce qui nous sera fort utile pour la suite.

$$f_0(x) = \begin{cases} 1 & \text{si } x = 0 \\ x + 1 & \text{si } 3x > 0 \\ f_0(x - 1) & \text{sinon} \end{cases} \tag{1.4}$$

La définition de f_0 est telle que $f_0(cn^K - 1) = cn^K$ et pour $x > cn^K$, $f_0(x) = cn^K$.

On peut utiliser cette astuce pour définir, en même temps que la phase d'initialisation, une fonction qui permet d'obtenir cn^K. Si $x < cn^K$, c'est-à-dire, $x = 0$ ou $x + x + x > 0$, on définit A et R'_A par :

$$A(x) = x \tag{1.1}$$
$$R'_A(x) = f(x) \tag{1.2}$$

Les autres fonctions valent 0 sur cet intervalle, sauf $f_0(x) = x + 1$. Pour l'instant, on n'utilise pas $f_0(x)$, qui de toute façon n'est pas prête à l'emploi.

Cette initialisation va permettre de récupérer les valeurs de R_A dans la deuxième partie du domaine.

15. On aura probablement besoin de l'opération $\times$ si on a besoin d'avoir la fonction $i \mapsto n^K$.
16. Je ne comprends pas si ce $c(\Gamma)$ est le même que celui qu'on vient d'instancier.
17. C'est l'opération $g, g' \mapsto g' \left[g^{\leftarrow}(x)\right]_x$

Sur le domaine $[\![cn^K, 2cn^K - 1]\!]$:

$$I(x) = \begin{cases} 1 & \text{si } x = cn^K \\ I'(x-1) & \text{sinon} \end{cases} \tag{1.1}$$

$$I'(x) = \begin{cases} i & \text{si } I(x) \text{ est de la forme IF}(A = B)\{I(i)\} \text{ ELSE } \{I(j)\} \text{ et } A(x) = B(x) \\ j & \text{si } I(x) \text{ est de la forme IF}(A = B)\{I(i)\} \text{ ELSE } \{I(j)\} \text{ et } A(x) \neq B(x) \\ I(x) & \text{si } I(x) \text{ est de la forme HALT} \\ I(x)+1 & \text{sinon} \end{cases} \tag{1.2}$$

$$A(x) = \begin{cases} 0 & \text{si } x = cn^K \\ A'(x-1) & \text{sinon} \end{cases} \tag{1.3}$$

$$A'(x) = \begin{cases} c & \text{si } I(x) \text{ est de la forme } A := c \\ A(x) * B(x) & \text{si } I(x) \text{ est de la forme } A := A + B \\ R_A(x) & \text{si } I(x) \text{ est de la forme } A := R_A \\ N(x) & \text{si } I(x) \text{ est de la forme } A := N \\ A(x) & \text{sinon} \end{cases} \tag{1.4}$$

$$B(x) = \begin{cases} 0 & \text{si } x = cn^K \\ B'(x-1) & \text{sinon} \end{cases} \tag{1.5}$$

$$B'(x) = \begin{cases} A(x) & \text{si } I(x) \text{ est de la forme } B := A \\ B(x) & \text{sinon} \end{cases} \tag{1.6}$$

$$N(x) = \begin{cases} n & \text{si } x = cn^K \\ N'(x-1) & \text{sinon} \end{cases} \tag{1.7}$$

$$N'(x) = \begin{cases} A(x) & \text{si } I(x) \text{ est de la forme } N := A \\ N(x) & \text{sinon} \end{cases} \tag{1.8}$$

$$R_A(x) = R'_A\left[A^{\leftarrow}(x)\right]_x \tag{1.9}$$

$$R'_A(x) = \begin{cases} B(x) & \text{si } I(x) \text{ est de la forme } R_A := B \\ R_A(x) & \text{sinon} \end{cases} \tag{1.10}$$

Notons que cn^K code le premier instant du calcul.

Les fonctions $I(x)$ et $I'(x)$ ne prennent qu'un nombre fini de valeurs. Ainsi, en écrivant *si $I(x)$ est de la forme $R_A := B$*, on écrit en fait *si $I(x) = i_1$ ou $I(x) = i_2$ ... ou $I(x) = i_m$* où $i_1, \ldots, i_m$ sont les numéros des instructions $R_A := B$ dans le programme de la RAM.

De plus, les opérations $I'(x-1)$ sont en réalité écrites $I\left[1^{\leftarrow}(x)\right]_x$ dans un vrai LSRS. Comme on l'a précédemment dit, on utilise R_A pour extraire les valeurs de sortie du calcul de M, donné par la valeur n' de N et le contenu $R(0), \ldots, R(n'-1)$, à la fin du calcul. Pour $2cn^K \leqslant x < 3cn^K$, le LSRS consiste en les équations suivantes :

$$A(x) = x - 2cn^K \tag{1.1}$$

$$R_A(x) = R'_A\left[A^{\leftarrow}(x)\right]_x \tag{1.2}$$

Les autres fonctions du LSRS sont définies par $h(x) = h(x-1)$ (les fonctions stationnent).

Il ne reste plus qu'à voir que les équations de S définissent les fonctions attendues pour décrire le calcul de M et que S produit la bonne sortie.

Soit s la RAM-structure, $1(-), n(-), id(-)$ [18]comme précédemment, et soient $I, A, B, N, R_A, I', A', B', N', R'_A$ des fonctions qui vérifient les équations de S.

Premièrement, il est clair que pour $a < c(s.n)^K$, on a :

$$A(a) = a \tag{1.1}$$

$$R'_A(a) = s.f(a) \tag{1.2}$$

$$I(a) = B(a) = N(a) = R_A(a) = I'(a) = A'(a) = B'(a) = N'(a) = 0 \tag{1.3}$$

Les valeurs des fonctions au point $c(s.n)^K$ décrivent la configuration de la RAM au début du calcul. De plus, la définition de A et R'_A sur la première partie du domaine permet de rendre compte du faire que M contient $s.f(i)$ dans le registre R_i.

18. On obtient la fonction $i \mapsto cn^K$ avec une astuce, cf cette note.

On montre facilement par induction sur t que la valeur des fonctions au temps $c(s.n)^K+t$ sont correctes. La partie la plus difficile réside dans le calcul de R_A. Remarquons premièrement que, puisque $A(a) < c(s.n)^K$ pour chaque a, l'initialisation sur la première partie du domaine assure que $A^{\leftarrow}(a) < a$ pour tout $a \in [\![cn^K, 2cn^K - 1]\!]$. Deux cas possibles :

- $c(s.n)^K \leqslant A^{\leftarrow}(a) < a$. Dans ce cas, il existe $b \in [\![c(s.n)^K, a-1]\!]$ tel que $A(a) = A(b)$; donc le registre courant R a déjà été visité pendant le calcul, et $A^{\leftarrow}(a)$ est la dernière étape où cela s'est produit ; ainsi $R'_A\left(A^{\leftarrow}(a)\right)$ donne la bonne valeur de $R_A(a)$.
- $A^{\leftarrow}(a) < c(s.n)^K$. Dans ce cas, il n'y a pas de $b \in [\![c(s.n)^K, a-1]\!]$ tel que $A(a) = A(b)$; donc le registre courant R n'a pas été visité pour le moment, et devrait toujours contenir la valeur de $f(A^{\leftarrow}(a))$ [19]. Par initialisation, il découle que $A^{\leftarrow}(a) = A(a)$ et $R'_A(A(a)) = f(A(a))$, comme voulu.

Enfin, on doit encore montrer que S définit correctement $\Gamma(s)$. Par définition de R_A et N sur la troisième partie du domaine du LSRS, pour chaque a, $R_A\left(2c(s.n)^K + a\right)$ contient les valeurs de du registre R_a à la fin du calcul. De plus, $N(3c(s.n)^K - 1)$ contient la valeur de N à la fin du calcul. Ainsi, avec une projection bien choisie, $\Gamma(s)$ peut être extraite des fonctions définies par S.

Ainsi, on a montré que S calcule correctement $\Gamma(s)$. Donc Γ est n^K-représentée par S.

□

19. Petite erreur de frappe dans l'article original ? Il y est écrit : $f(A(a))$.

Chapitre 2

LSRS à arité multiple

2.1 Introduction

Avant toute chose, il peut être bon de rappeler à quoi ressemblent les LSRS. [3] [1]. Cependant, pour les besoins du présent chapitre, nous allons ajouter une troisième forme d'opération.

Définition 10 (LSRS)**.** *Soit F un ensemble de symboles de fonctions (dites* fonctions de base*), soient $f_1, \dots, f_k$ des symboles de fonctions qui n'apparaissent pas dans F. Pour $i \leqslant k$, notons $F_i = F \cup \{f_1, \dots, f_i\}$.*

*Un LSRS (*Linear Simultaneous Recursion Scheme*) S sur $f_1, \dots, f_k$ et F est une suite de k équations $(E_i)_{i_\in k}$ dont chacune est de l'une des trois formes suivantes :*

- *(opération)* $f_i(x) = g(x) * g'(x)$ *où* $g, g' \in F_{i-1}$ *et* $* \in \{+, -, \times\}$ [1]
- *(récursion)* $f_i(x) = g'\left[g^{\leftarrow}(x)\right]_x$ *où* $g' \in F_k$ *et* $g \in F_{i-1}$
- *(composition)* $f_i(x) = g'\left[g(x)\right]_x$ *où* $g' \in F_k$ *et* $g \in F_{i-1}$

Remarque 2. L'ajout de cette opération facilitera les calculs qui suivront. Elle ne rend pas le LSRS plus puissant [1].

Le but de ce chapitre est d'étendre les LSRS définis dans [1] et utilisés plus haut aux fonctions d'arité > 1. Remarquons dans un premier temps que si l'on définit un LSRS avec des fonctions d'arité a, où a est le même pour chaque fonction, alors l'ordre lexicographique permet de facilement définir un LSRS équivalent d'arité 1, ce qui n'apporte pas grand-chose par rapport à ce dont nous avons déjà parlé.

Ici, on ne va pas considérer de LSRS dont toutes les fonctions sont de même arité. Soit $a > 1$, on suppose que le LSRS est composé de fonctions f_i d'arité $r_i \leqslant a$, où $i \in k$, et telles que $r_i = a$ pour au moins un $i \in k$. On qualifie de a-LSRS un tel LSRS. On notera LSRS les systèmes utilisés dans le chapitre précédent, qui sont en fait des 1-LSRS.

Dans la suite, on notera $(\leqslant a)$-uplet pour parler d'un n-uplet où $n \leqslant a$.

2.2 Les ennuis commencent

2.2.1 Bon ordre sur les $(\leqslant a)$-uplets

Dans un LSRS, l'ordre est très important ; que ce soit au niveau de l'ordre des fonctions ou des variables. De manière générale, dans un 1-LSRS, $f(x)$ ne peut être défini avec $g(y)$ que si $g(y)$ a été calculé avant, que ce soit parce que $y < x$ ou, si $x = y$, parce que g est définie avant elle dans le LSRS. Pour reproduire l'importance de l'ordre sur des fonctions d'arité multiple, et ne pas trahir la définition originelle du LSRS, on peut utiliser un ordre lexicographique sur les $(\leqslant a)$-uplets tels que $\bar{x} <_{\text{naïf}} \bar{y} \Leftrightarrow |\bar{x}| < |\bar{y}|$ ou $\bar{x} <_{\text{lex}} \bar{y}$.

Cela revient à calculer, dans l'ordre, d'abord les fonctions d'arité 1, puis ensuite toutes les fonctions d'arité 2, etc. jusqu'à calculer les fonctions d'arité a. Le problème de cette définition est que les fonctions d'arité a' ne peuvent faire appel qu'aux fonctions d'arité $< a'$, ce qui ressemble à une contrainte inutile [2].

Nous avons choisi un ordre moins naturel, mais qui est un bon ordre, permet de faire des projections (récupérer des éléments d'un a'-uplet, sauf au moins un, et les utiliser comme argument d'une autre fonction), et permet de calculer tour à tour des fonctions d'arité 1, puis 2, etc., puis revenir aux arités 1, 2...

1. Les opérations peuvent varier, il est dit dans [1] que l'on peut choisir n'importe quelle opération binaire calculable en temps linéaire sur machine de Turing, voire la multiplication. Nous verrons d'ailleurs par la suite que, pour des raisons de facilité, nous aurons besoin de la multiplication.

2. Il y avait de plus une autre raison, mais je ne m'en souviens plus au moment où j'écris ces lignes. C'est apparemment un bon ordre (*tout ensemble non vide a un plus petit élément*), mais il y avait un contre-argument qui nous avait fait chercher un autre ordre. Parce qu'on atteint très vite les bornes du domaine ?

Définition 11. *On définit $<$ sur les $(\leqslant a)$-uplets par :*

$$\bar{x} < \bar{y} \Leftrightarrow \begin{cases} \max(\bar{x}) < \max(\bar{y}) \\ \textit{ou } \max(\bar{x}) = \max(\bar{y}) & \textit{et } |\bar{x}| < |\bar{y}| \\ \textit{ou } \max(\bar{x}) = \max(\bar{y}) & \textit{et } |\bar{x}| = |\bar{y}| \quad \textit{et } \bar{x} <_{lex} \bar{y} \end{cases}$$

Si toutes les arités $\leqslant a$ ne sont pas permises (par exemple, si notre LSRS ne contient que des fonctions d'arité 1 et 3, mais pas 2), alors on ampute cet ordre des n-uplets correspondant.

Exemple 12. *Pour $a = 3$, avec les arités $1, 2, 3$:* $(0) < (0,0) < (0,0,0) < (1) < (0,1) < (1,0) < (1,1) < (0,0,1) < (0,1,0) < (0,1,1) < (1,0,0) < (1,0,1) < (1,1,0) < (1,1,1) < (2) < (0,2) < \dots$.

Pour $a = 3$, avec les arités $1, 3$: $(0) < (0,0,0) < (1) < (0,0,1) < (0,1,0) < (0,1,1) < (1,0,0) < (1,0,1) < (1,1,0) < (1,1,1) < (2) < (0,0,2) < \dots$.

Remarque 3. Quelques remarques informelles :

- Le premier $(\leqslant a)$-uplet d'un max m donné est le 1-uplet (m).
- Le premier $(\leqslant a)$-uplet d'une arité r donnée, à un max m donné, est le r-uplet $(0, \dots, 0, m)$.

2.2.2 Propriétés combinatoires

Proposition 13. *Soit $\bar{x}$ un $(\leqslant a)$-uplet. On suppose que toutes les arités sont possibles. Notons $m = \max(\bar{x})$ et $r = |\bar{x}|$.*

Le bon ordre $<$ sur les $(\leqslant a)$-uplets vérifie les égalités suivantes :

1. $card(\{\bar{y} < \bar{x} | \max(\bar{y}) < \max(\bar{x})\}) = \sum_{i=1}^{a} m^i$ [3] ;
2. $card(\{\bar{y} < \bar{x} | \max(\bar{y}) = \max(\bar{x}) \wedge |\bar{y}| < |\bar{x}|\}) = \sum_{i=1}^{r} \left((m+1)^i - 1\right)$;
3. $card(\{\bar{y} < \bar{x} | \max(\bar{y}) = \max(\bar{x}) \wedge |\bar{y}| = |\bar{x}| \wedge \bar{y} <_{lex} \bar{x}\}) = \sum_{i=1}^{r-1} x_i m^{r-i}$. *PROBABLEMENT FAUX parce qu'il faut être sûr de garder un m quelque part dans le r-uplet.*

Démonstration. 1. L'entier a étant fixé, pour un max m donné, on peut construire un $(\leqslant a)$-uplet en choisissant une arité $i \leqslant a$ et i composantes, toutes des entiers $< m$. Pour i fixé, on peut faire m^i i-uplets, d'où au total $\sum_{i=1}^{a} m^i$.

2. Pour construire un tel $\bar{y}$, il faut choisir son arité $i < r$, puis le nombre de composantes que l'on fixe à m ; le reste des composantes est libre et $< m$, d'où le résultat.

3. Je dois y réfléchir. □

Corollaire 14. *Le rang de $\bar{x}$ est* $\left(\sum_{i=1}^{a} m^i\right) + \left(\sum_{i=1}^{r} \left((m+1)^i - 1\right)\right)$ + *quelque chose.*

2.3 Déroulement d'un calcul de a-LSRS

Au lieu d'incrémenter x de façon naturelle en $x+1$, ici, on compte passer de $\bar{x}$ à son successeur dans cet ordre. Notons-le $s(\bar{x})$. Ainsi, le calcul d'un a-LSRS se déroule de la façon suivante :

- $\bar{x} = (0)$: calcul de toutes les fonctions d'arité 1 en (0), comme dans un LSRS normal. Dans cette première étape, les fonctions ne peuvent bien sûr pas faire référence aux fonctions d'arité supérieure, car elles ne sont même pas encore définies ; leur évaluation doit respecter les règles du LSRS énoncées plus haut.
- $\bar{x} = (0,0) = s((0))$: calcul de toutes les fonctions d'arité 2 en $(0,0)$. Ces fonctions peuvent faire appel aux valeurs des fonctions d'arité 1 en 0. Une fonction f_i d'arité 2 peut aussi faire référence à des fonctions d'arité 2 en $(0,0)$, à condition qu'elles aient déjà été évaluées ; c'est-à-dire, à condition qu'elles soient définies dans des équations E_j telles que $j < i$;
- $s(\bar{x})$:
  - Si $r = |\bar{x}| = |s(\bar{x})|$ alors on calcule à nouveau les fonctions d'arité r pour $s(\bar{x})$; ces fonctions peuvent faire appel aux résultats des fonctions d'arité plus petite.
  - Si $r = |\bar{x}| < |s(\bar{x})| = r'$ alors on évalue les fonctions d'arité r' en $s(\bar{x})$; idem, ces fonctions peuvent faire appel aux résultats des fonctions d'arité plus petite.
  - Si $|s(\bar{x})| = 1 < a = |\bar{x}|$ alors on évalue les fonctions d'arité 1 en $s(\bar{x})$; ces fonctions d'arité 1 peuvent faire référence aux résultats des fonctions d'arité plus grande, à condition qu'ils aient été calculés au tour précédent.

Une fois l'ordre compris, il faut voir comment interpréter les opérations typiques du LSRS : addition, soustraction, multiplication, récursion et composition.

3. J'ai mis -1 ici au début, et j'avais l'air convaincu, mais je ne sais plus pourquoi...

Définissons le a-LSRS Ce qui suit est la piste principale que nous suivons en ce moment. Le LSRS pourrait être élargi de bien des manières, mais c'est celle-ci qui nous semble la plus intéressante et la plus naturelle, une fois que l'on a choisi comment ordonné les $(\leqslant a)$-uplets.

Soit $a \in \mathbb{N}$. Le symbole "$<$" renvoie selon les cas à l'ordre sur les entiers ou à l'ordre juste au-dessus. Pour des raisons de lisibilité, on notera $\bar{x'} \ll \bar{x}$ pour dire : $\bar{x'} < \bar{x}$, $\left|\bar{x'}\right| < |\bar{x}|$ et $\forall j\ \exists j'\ x'_j = x_{j'}$ [4].

Définition 15 (a-LSRS)**.** *Soit F un ensemble de symboles de fonctions de base. Soient $f_1, \dots, f_k$ de nouveaux symboles de fonctions n'apparaissant pas dans F, d'arités respectives $1 \leqslant r_1 \leqslant r_2 \leqslant \cdots \leqslant r_k = a$* [5]*. On note $F_i = F \cup \{f_j | r_j = r_i \text{ et } j < i\}$, $F'_i = F \cup \{f_j | r_j = r_i\}$, et $G_i = F \cup \{f_j | r_j < r_i\}$* [6]*.*

Un a-LSRS S sur F et $f_1, \dots, f_k$ est une suite d'équations $E_1, \dots, E_k$ où chaque E_i est de l'une des formes suivantes :

- *(opération) $f_i(\bar{x}) = A * B$ où $* \in \{+, -, \times\}$ et A, B sont de la forme suivante :*
  - *$g(\bar{x})$, avec $g \in F_i$;*
  - *$g\left(\bar{x'}\right)$, avec $g \in G_i$, c et $\bar{x'} \ll \bar{x'}$.*
- *(récursion) $f_i(\bar{x}) = g'\left[g^{\leftarrow}(\bar{x'})\right]_{\bar{x'}}$, où arité$(g)$ = arité(g') et l'un des deux cas suivants se réalise :*
  - *Soit $\bar{x'} = \bar{x}$, et dans ce cas $g \in F_i$ et $g' \in F'_i$;*
  - *Soit $\bar{x'} \ll \bar{x}$ et dans ce cas $g, g' \in G_i$.*
- *(composition)* [7] *$f_i(\bar{x}) = g'[g_1(\bar{x_1}), \dots, g_r(\bar{x_r})]_{\bar{x}}$, où arité$(g') = r$, et pour chaque $j \in r$, l'un des cas suivants se réalise :*
  - *Soit $\bar{x_j} = \bar{x}$, et dans ce cas $g_j \in F_i$;*
  - *Soit $\bar{x_j} \ll \bar{x}$ et dans ce cas $g_j \in G_i$.*

  De plus :
  - *Soit $r = |\bar{x}|$, et dans ce cas $g' \in F_i$* [8] *;*
  - *Soit $r < |\bar{x}|$ et dans ce cas $g' \in G_i$.*

Remarque 4. Par défaut, on suppose, une nouvelle fois, que l'ensemble des fonctions de base contient par défaut les symboles $n(-)$ et $1(-)$ pour toutes les arités possibles, ainsi que les $\pi^i_j(-)$, projections récupérant la i-ième composante d'un j-uplet. On peut de plus rajouter $a(-)$, permettant de récupérer a, quoique cette fonction est facilement calculable par un LSRS.

On doit aussi adapter les notions de structures de RAM et de fonctions de RAM. On garde le même a que pour le a-LSRS.

Définition 16 (RAM-structure)**.** *Soit t un a-type, c'est-à-dire une signature fonctionnelle contenant des fonctions d'arité $\leqslant a$ et dont au moins un symbole est d'arité a.*

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

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

On notera $s.n, s.C, s.f$ les composantes n, C, f de s.

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

Définition 17 (Fonction de RAM)**.** *Soient t_1, t_2 des a_1, a_2-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 [9]*.*

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

4. Autrement dit : $\bar{x'}$ est obtenu à partir de $\bar{x}$ en récupérant ses composantes, en les mélangeant, en les dupliquant, mais en veillant à ce que $\left|\bar{x'}\right| < |\bar{x}|$. Par exemple, pour $\bar{x} = (x_1, x_2, x_3)$, les $\bar{x'}$ pourraient être (x_1, x_2), (x_3, x_1) ou (x_3, x_3).

5. L'ordre entre des fonctions de même arité a de l'importance, mais pas entre des fonctions d'arités différentes. On peut donc considérer que les équations sont ordonnées par l'arité de la fonction qu'elles définissent.

6. F_i est l'ensemble des symboles des fonctions qui ont la même arité que f_i mais qui sont définies avant f_i.
F'_i est l'ensemble des symboles des fonctions qui ont la même arité que f_i, qu'elles soient définies avant ou après f_i.
G_i est l'ensemble des symboles des fonctions d'arité strictement inférieure à celle de f_i.

7. Cette opération n'est pas forcément nécessaire ; je l'avais rajoutée vraisemblablement en me trompant quand à son usage. Je reviendrai probablement dessus plus tard.

8. Si $\bar{x} < (g_1(\bar{x_1}), \dots, g_r(\bar{x_r}))$ il faut définir une valeur par défaut à $g'[g_1(\bar{x_1}), \dots, g_r(\bar{x_r})]_{\bar{x}}$. Dans le LSRS normal, on se contente de renvoyer x. On peut ici renvoyer une composante par défaut, par exemple x_1.

9. On rappelle que "c-borné" ne concerne que la structure par rapport à sa propre taille ; ici on ne compare pas la taille de l'entrée et de la sortie

Définition 18 (RAM représentée par a-LSRS)**.** *Soient t_1, t_2 respectivement un a_1-type et un a_2-type. Soit Γ une (t_1, t_2)-fonction de RAM.*

Soit S un a-LSRS pour $f_1, \dots, f_k$ sur F_{t_1}.

On dit que Γ est 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 $f_1, \dots, f_k : cs.n \times \cdots \times cs.n \to (cs.n)^a$ telles que $\Gamma(s) = P((s.n)^a, S(s))$ ($S(s)$ est la structure définie par le a-LSRS).

A priori, les a-LSRS prennent en entrée des structures dont l'arité maximale est au plus a, car les définitions que l'on a données ne permettent pas d'utiliser des fonctions dont l'arité est plus grande que a.

2.4 Lien entre les deux notions de LSRS

L'idée principale de ce chapitre est de chercher un lien entre les LSRS et les a-LSRS.

Conjecture 5. *Soit Γ une (t_1, t_2)-fonction de RAM, où t_1 est un 1-type et t_2 est un a-type.*
Γ est représentable par un a-LSRS ssi Γ est n^a-représentable par un LSRS.

Pour prouver cette conjecture, nous allons avoir besoin de coder l'entrée.

Lemme 19. *Il existe un 1-LSRS S qui, à une variable x, associe le $(\leqslant a)$-uplet $\bar{x}$ de rang x.*

Démonstration. Pour simplifier, on va de nouveau se placer dans le cas où toutes les arités $\leqslant a$ sont représentées [10]. On va se servir du codage du rang donné en proposition 13, montrer qu'on peut le décoder par LSRS.

Calcul du max m. Considérons le LSRS suivant :

$$f_1(x) = \begin{cases} 0 & \text{si } x < a \\ 1 & \text{si } x = a \\ f_1(x-1) & \text{si } x < f_{4a-1}(x-1) \\ f_1(x-1)+1 & \text{si } x = f_{4a-1}(x-1) \end{cases} \tag{2.1}$$

$$f_{i_1+1}(x) = f_{i_1}(x) \times f_1(x) \tag{2.2}$$

$$f_{a+1}(x) = f_1(x) + f_2(x) \tag{2.3}$$

$$f_{a+i_2+1}(x) = f_{a+i_2}(x) + f_{i_2+2}(x) \tag{2.4}$$

$$f_{2a}(x) = f_1(x) + 1 \tag{2.5}$$

$$f_{2a+i_3+1}(x) = f_{2a+i_3}(x) \times f_{2a}(x) \tag{2.6}$$

$$f_{3a}(x) = f_{2a}(x) + f_{2a+1}(x) \tag{2.7}$$

$$f_{3a+i_4+1}(x) = f_{3a+i_4}(x) + f_{2a+i_4+2}(x) \tag{2.8}$$

On rappelle que $f_1(x-1) = f_1\left[1^{\leftarrow}(x)\right]_x$ et que la distinction de cas ne rend pas le LSRS plus puissant [1].
Expliquons les équations.
(2.1) est censée calculer m, donc le max courant. Sachant cela, les équations suivantes coulent de source :

- (2.2) calcule m^{i_1} pour $i_1 \in [1, a]$. On a aussi $f_a(x) = m^a$.
- (2.3) calcule $\sum_{i=1}^{2} m^i$ et (2.4) calcule $\sum_{i=1}^{i_2+2} m^i$ pour $i_2 + 2 \leqslant a$. On a ainsi $f_{2a-1}(x) = \sum_{i=1}^{a} m^i$, soit le rang du 1-uplet (m).
- Les équations suivantes sont sur le même schéma, si ce n'est qu'elles travaillent sur $m+1$ au lieu de m. On a $f_{3a-1}(x) = (m+1)^a$ et $f_{4a-1}(x) = \sum_{i=1}^{a}(m+1)^i$, donc $f_{4a-1}(x)$ contient le rang du 1-uplet du max suivant, à savoir $(m+1)$.

Ceci étant explicité, étudions la définition de $f_1(x)$.
$f_1(x)$ s'initialise naturellement en 0. On parcourt alors tous les $(\leqslant a)$-uplets $(0, \dots, 0)$ jusqu'à l'arité a, donc on énumère les a premiers $(\leqslant a)$-uplets, et donc l'élément de rang a a bien un max égal à 1 : $f_1(x)$ passe à 1. Quand le calcul avance, x finit par atteindre le rang du max suivant, que l'on calcule par avance dans $f_{4a-1}(x)$. Une fois que ce rang est atteint, on incrémente $f_1(x)$; le reste des équations se met aussi à jour, ce qui assure que $f_{4a-1}(x)$ est toujours le rang du max suivant. La distinction de cas de $f_1(x)$ est donc bien complète.

10. Si certaines arités ne sont pas représentées, alors il suffit de retirer les quelques équations qui y sont associées et de modifier, le cas échéant, les initialisations de certaines fonctions.

Calcul de l'arité courante Le calcul de l'arité du $(\leqslant a)$-uplet de rang x s'effectue avec la même idée. Pour des raisons de lisibilité, le système présenté ne sera pas un véritable LSRS mais on laisse le lecteur se convaincre qu'il est facile de réécrire ce système sous une véritable forme de LSRS en s'inspirant de ce qui a été fait au-dessus, quitte à remanier la numérotation des fonctions.

$$f_{4a}(x) = \begin{cases} 1 & \text{si } x = 0 \\ f_{4a}(x-1) & \text{si } x < f_{8a+2}(x-1) \\ 1 & \text{si } x = f_{4a-1}(x-1) \\ f_{4a}(x-1)+1 & \text{si } x = f_{8a+2}(x-1) \end{cases} \tag{2.9}$$

$$f_{i_1+4a}(x) = \begin{cases} (m+1)^{i_1}-1 & \text{si } 1 \leqslant i_1 \leqslant f_{4a}(x) \leqslant a \\ 0 & \text{sinon} \end{cases} \tag{2.10}$$

$$f_{i_2+5a}(x) = \sum_{i=1}^{i_2} f_{i+4a}(x) \tag{2.11}$$

$$= \sum_{i=1}^{\min(f_{4a}(x),i_2)} \left((m+1)^i - 1\right)$$

$$f_{6a+1}(x) = \begin{cases} f_{4a}(x)+1 & \text{si } f_{4a}(x) < a \\ 1 & \text{sinon} \end{cases} \tag{2.12}$$

$$f_{i_3+6a+1}(x) = \begin{cases} (m+1)^{i_3}-1 & \text{si } 1 \leqslant i_3 \leqslant f_{6a+1}(x) \leqslant a \\ 0 & \text{si } f_{6a+1}(x) = 1 = i_3 \\ 0 & \text{sinon} \end{cases} \tag{2.13}$$

$$f_{i_4+7a+1}(x) = \sum_{i=1}^{i_4} f_{i+6a+1}(x) \tag{2.14}$$

$$= \begin{cases} \sum_{i=1}^{\min(f_{6a+1}(x),i_4)} \left((m+1)^i - 1\right) & \text{si } 1 < f_{6a+1}(x) \\ 0 & \text{sinon} \end{cases}$$

$$f_{8a+2}(x) = f_{8a+1}(x) + f_{4a}(x) \tag{2.15}$$

On va un peu plus vite : les équations (2.10), (2.11), (2.13) et (2.14) sont des calculs intermédiaires aux résultats suivants : si $f_{4a}(x)$ calcule l'arité courante, alors $f_{6a+1}(x)$ calcule l'arité suivante, et $f_{8a+2}(x)$ calcule le rang du premier $f_{6a+1}(x)$-uplet. Enfin, $f_{4a}(x)$ vérifie le résultat de $f_{8a+2}(x-1)$ pour s'adapter : si $x = f_{8a+2}(x)$ alors on a atteint le seuil qui détermine qu'on est passé à l'arité suivante. Comme $f_{4a}(x)$ se met à jour, $f_{8a+2}(x)$ se met aussi à jour pour donner le rang du premier terme d'arité suivante, donc x ne dépasse jamais $f_{8a+2}(x)$. Les autres cas de la distinction de cas sont triviaux : on retourne à 1 quand la fonction $f_{4a-1}(x-1)$ (cf. plus haut) indique qu'on a changé de max courant.

Calcul des cordonnées. Ici, c'est rigolo. Cette partie de la démonstration est entièrement dépendante de la sous-section 2.2.2, pour laquelle il me manque la troisième partie.

Je peux néanmoins glisser quelques idées qu'il faudra prendre en compte, après avoir rappelé que le max courant est noté m :

- Puisque l'arité est changeante, certaines fonctions devront renvoyer une valeur par défaut, par exemple si leur indice est plus grand que l'arité courante.
- Le premier terme d'une arité donnée est $(0, \ldots, 0, m)$ et le suivant est soit $(0, \ldots, 1, m)$ si $m \neq 1$ soit $(0, \ldots, 1, 0)$ si $m = 1$.
- Quand on parcourt les termes d'une arité fixée, à part le fait de garder le max quelque part, le reste des calculs revient à une énumération de mots de longueur donnée, dans un alphabet de taille m.

□

Une fois qu'on a les coordonnées, le deuxième problème qui se pose est l'utilisation des coordonnées partielles, ce que transcrivait la notation $\bar{x'} \ll \bar{x}$: $\bar{x'}$ est obtenu en prenant des coordonnées à $\bar{x}$, en les mélangeant, éventuellement en dupliquant certaines, tant que $\left|\bar{x'}\right| < |\bar{x}|$. Une fois qu'on a le $(\leqslant a)$-uplet $\bar{x}$, il est facile de récupérer $\bar{x'}$. Mais il faut ensuite convertir $\bar{x'}$ en son rang x' afin que le LSRS d'arité 1 puisse s'en servir.

Le calcul du rang x' de $\bar{x'}$ est entièrement dépendant de la troisième partie de la sous-section 2.2.2. Admettons qu'on ait cette fonction qui à un $(\leqslant a)$-uplet associe son rang. Alors on a le lemme suivant :

Lemme 20. *Supposons que g_1, g_2 sont des symboles de fonctions utilisés ou définies dans un a-LSRS, et sont simulées par $\hat{g_1}, \hat{g_2}$ dans un LSRS.*

Alors l'opération $f(\bar{x}) = g_1(\bar{x_1}) * g_2(\bar{x_2})$, pour $* \in \{+, -, \times\}$ d'un a-LSRS est simulable par LSRS.

Démonstration. Tout d'abord, présentons quelques notations. Notons $\text{sub}_j(x)$ l'application qui à un code x associe x_j, code du $(\leqslant a)$-uplet $\bar{x_j} \ll \bar{x}$.

On a dû admettre l'existence de $\text{sub}_j(x)$ plus haut, mais j'ai bon espoir qu'elles soient effectivement définissables par un LSRS.

Dans le LSRS d'arité 1, on va devoir ajouter la composition bornée $g, g' \mapsto (x \mapsto g'[g(x)]_x)$. Cet ajout ne rend pas le LSRS plus puissant [1].

Le LSRS suivant [11]simule $f(\bar{x}) = g_1(\bar{x_1}) * g_2(\bar{x_2})$:

$$\begin{aligned} f_1(x) &= \hat{g_1}[\text{sub}_1(x)]_x && (2.16)\\ f_2(x) &= \hat{g_2}[\text{sub}_2(x)]_x && (2.17)\\ f^1(x) &= f_1(x) * f_2(x) && (2.18) \end{aligned}$$

□

Remarque 5. Avant de traiter l'opération suivante, il faut apporter une précision. Soit le 2-LSRS suivant :

$$\begin{aligned} f_1(x) &= n(x) + 1(x) && (2.19)\\ f_2(x,y) &= f_1(x) + 1(y) && (2.20)\\ f_3(x,y) &= f_1(x) + id(y) && (2.21) \end{aligned}$$

Supposons-le simulé par un LSRS S sur $\hat{f_1}, \hat{f_2}, \hat{f_3}$ (avec éventuellement d'autres fonctions, mais elles ne sont pas importantes pour la remarque).

L'ordre est : $(0) < (0,0) < (1) < (0,1) < (1,0) < (1,1) < (2) \ldots$. Soit x un $(\leqslant 2)$-uplet, notons $r(x)$ son rang. La définition donnée ici du déroulement d'un a-LSRS suppose, en quelque sorte, qu'une fois que $f_1(0)$ est calculé, on avance d'un pas dans l'ordre et $f_1(0)$ reste *figé* en attendant qu'on ait calculé $f_2(0,0)$ et $f_3(0,0)$. Dans la simulation par un LSRS, chaque fonction renvoie constamment un résultat ; ici, que renvoie $\hat{f_1}(r(0,0))$? Une idée pourrait être simplement de renvoyer son dernier résultat. L'idée est que $\hat{f_1}$ effectue son calcul sur $r(\bar{x})$ tant que $|\bar{x}| = \text{arité}(f_1)$, puis renvoie son dernier résultat tant que l'arité n'est pas la bonne. Il faut donc que chaque fonction ait accès à l'arité de la fonction qu'elle simule, ce qui peut se faire en rajoutant des fonctions auxiliaires, incluses dans le LSRS.

En conclusion, si $f(\bar{x}) = \sigma(\bar{x})$ est une équation de a-LSRS, avec f d'arité r, alors sa simulation est de la forme :

$$\begin{aligned} \text{arité}_1(x) &= 1(x) && (2.22)\\ &\ldots\\ \text{arité}_a(x) &= \text{arité}_{a-1}(x) + 1(x) && (2.23)\\ &\ldots\\ \text{a\_c}(x) &= \text{arité courante} && (2.24)\\ &\ldots\\ \hat{f}(x) &= \begin{cases} \hat{\sigma}(\bar{x}) & \text{si a\_c}(x) = \text{arité}_r(x) \\ \hat{f}(x-1) & \text{sinon} \end{cases} && (2.25) \end{aligned}$$

Dans la suite, pour des raisons de lisibilité, on évitera ce genre d'écriture, et on notera simplement la simulation sous sa forme $\hat{f}(x) = \hat{\sigma}(\bar{x})$.

Lemme 21. *Supposons que g_1, g_2 sont des symboles de fonctions utilisés ou définies dans un a-LSRS, et sont simulées par $\hat{g_1}, \hat{g_2}$ dans un LSRS.*

Alors l'opération $f(x) = g_1\left[g_2^{\leftarrow}(\bar{x'})\right]_{\bar{x}}$ d'un a-LSRS est simulable par un LSRS.

Démonstration. La remarque donnée précédemment pose un gros problème : elle ruine le *equal-predecesor*. □

11. La numérotation est disjointe de celles précédemment données.

Bibliographie

[1] E. Grandjean and T. Schwentick. Machine-independent characterizations and complete problems for deterministic linear time. *SIAM Journal on Computing*, 32(1) :196–230, 2003. cited By 8.

[2] Etienne Grandjean. Invariance properties of rams and linear time. *Computational Complexity*, 4(1) :62–106, 1994.

[3] Thomas Schwentick. Algebraic and logical characterizations of deterministic linear time classes. In *Proceedings of the 14th Annual Symposium on Theoretical Aspects of Computer Science*, STACS '97, pages 463–474, London, UK, UK, 1997. Springer-Verlag.