Documents de travail

pour l'oral d'informatique des ENS

Pierre-Louis Curien et Pascal Koiran

Décembre 2000

ENS Lyon

ENS Ulm

Préface

On trouvera rassemblés ici les documents qui nous ont servi de base de travail pour faire passer l'épreuve orale d'Informatique commune aux ENS Lyon et Ulm en 2000. (Rappelons que cette épreuve consiste en une interrogation au tableau d'une durée de 45 minutes, sans préparation préalable et sans partie "pratique" (programmation sur machine).

Ce recueil fait suite à celui correspondant à l'oral Cachan/Lyon/Ulm de 1999, réalisé par P.-L. Curien et Y. Robert (`http://www.ens-lyon.fr/~yrobert`). Les fichiers postcript sont disponibles sur les pages Web des auteurs.

Ce qui suit est repris de la préface du premier recueil. Chaque sujet traité explore un thème de manière approfondie, et partant, constitue un ensemble de questions et réponses bien plus ambitieux que le sujet véritablement posé à l'oral (lequel se limite souvent aux premières questions du document). Nous conseillons vivement à nos lecteurs de chercher par eux-mémes les réponses aux questions avant de lire la solution, toujours incluse dans le texte.

De nombreux sujets sont rédigés à l'aide d'un formalisme cher aux sémanticiens, qui surprendra peut-étre le lecteur au premier abord. Mais nous espérons le convaincre de l'élégance et de la rigueur des méthodes de dérivation formelles. La notation

$$\frac{P_1 \quad \cdots \quad P_n}{Q}$$

signifie: si les prédicats $P_1, \ldots, P_n$ sont vrais, alors on peut conclure (ou inférer – on parle de règle d'inférence) que le prédicat Q est vrai. Une absence de numérateur signifie que la conclusion est toujours vraie – on parle alors d'axiome. On utilise également un formalisme de réécriture pour décrire de nombreux algorithmes. Ainsi, on écrira

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \cdots \rightarrow S_n \rightarrow !V ,$$

où chaque S_i codifie un état (par exemple l'ensemble des valeurs des différentes variables, tableaux et autres structures de données utilisées par l'algorithme). La séquence de symboles ci-dessus se lit ainsi: à partir d'un état initial S_0, l'algorithme passe successivement par les états $S_1, \ldots, S_n$ pour s'achever avec un résultat V (la valeur calculée par l'algorithme). Le point d'exclamation souligne que l'algorithme a terminé. On note $\rightarrow^*$ la fermeture réflexive et transitive de $\rightarrow$ (par exemple, $S_0 \rightarrow^* !V$ ci-dessus).

Le texte a une forme très libre, parfois inachevée, et contient inévitablement quelques coquilles ou (plus grave!) des erreurs. Nous remercions par avance nos lecteurs de nous signaler celles-ci (par exemple par courrier électronique). Nous espérons que ce recueil sera utile aux candidats, à leurs professeurs, et à nos collègues. Bien entendu, les documents présentés ici reflètent le style, et pour partie les préférences, de leurs auteurs. Nous croyons cependant avoir été fidèles à la tradition des épreuves aux ENS, qui privilégient la réflexion, la curiosité, l'initiative, et l'originalité. Bon travail!

Décembre 2000

Pierre-Louis Curien, `Pierre-Louis.Curien@pps.jussieu.fr`,
`http://www.pps.jussieu.fr/~curien`
Pascal Koiran, `Pascal.Koiran@ens-lyon.fr`,
`http://www.ens-lyon.fr/~ykoiran`

Contents

1 L'addition **7**

2 Calcul d'entiers **11**

3 Théorème de Spira **15**

4 Couplage maximum **19**

5 Mariages stables **25**

6 Mariage randomisé **31**

7 Processus séquentiels et bisimulation **37**

8 Graphes série-parallèles **43**

9 Unification **49**

10 Diagrammes 2-dimensionnels **59**

11 Algèbres combinatoires **67**

12 Sémantique des variables locales **73**

1 L'addition

Quelques rappels sur les circuits booléens: ce sont des graphes orientés sans cycles; il y a des portes de calcul de trois types: les portes $\vee$ et $\wedge$ ont deux entrées, les portes $\neg$ une seule, et elles effectuent les opérations qu'on imagine. Les entrées du circuit sont placées aux sommets de degré entrant 0, les sorties sont recueillies aux sommets de degré sortant 0.

(A1) Ecrire l'algorithme d'addition de deux nombres entiers naturels A et B donnés par leur représentation binaire sur n bits.

(A2) Décrire le circuit booléen correspondant, évaluer le nombre de portes du circuit et le temps de calcul nécessaire pour produire $A + B$ à partir de A et B.

(A3) Le circuit obtenu à la question précédente est de profondeur proportionnelle à n. Comment transformer ce circuit pour diminuer la constante de proportionnalité ? (Indication: penser aux portes de négation.)

Preuve. (A1) Voici l'algorithme de l'école primaire. $A(0)$, $B(0)$ sont les chiffres de poids faibles de A et B. La somme C est stockée dans le tableau $C(0..n)$.

```
r := 0
pour i de 0 a n − 1 faire
        s := A(i) + B(i) + r
        si s ≥ 2
                alors
                        r := 1
                        si s = 2 alors C(i) := 0 sinon C(i) := 1
                sinon
                        si s = 0 alors C(i) := 0 sinon C(i) := 1
finpour
C(n) := r
```

(A2) Dans l'algorithme précédent on peut identifier comme "cellule de base" l'additionneur à un bit. Celui-ci prend en entrée 3 bits a, b, re (retenue entrante) et produit en sortie les 2 bits de la somme $a + b + re$. Le bit de poids fort $rs(a, b, re)$ est à 1 si et seulement si au moins deux des trois entrées sont égales à

1. Le bit de poids faible $s(a, b, re)$ est à 1 si et seulement si le nombre d'entrées à 1 est impair. On peut donc écrire par exemple:

$$\begin{aligned} rs(a, b, re) &= (a \wedge b) \vee (a \wedge re) \vee (b \wedge re) \\ s(a, b, re) &= (a \wedge b \wedge re) \vee (a \wedge \neg b \wedge \neg re) \vee (\neg a \wedge b \wedge \neg re) \vee (\neg a \wedge \neg b \wedge re)\ . \end{aligned}$$

Le circuit désiré s'obtient en mettant bout à bout n cellules de base. Plus précisément, si r_i désigne la valeur de la retenue au début de l'itération i de l'algorithme, on a $C(n) = r_n$ et pour $i < n$, $C(i) = s(A(i), B(i), r_i)$ et $r_{i+1} = rs(A(i), B(i), r_i)$. Chaque cellule de base comportant 22 portes (11 $\wedge$, 5 $\vee$ et 6 $\neg$), ceci nous donne un circuit de $22n$ portes. (Remarque: on peut économiser des portes dans la première cellule de base puisque la retenue entrante est toujours nulle. Plus précisément, on a $rs(a, b, 0) = a \wedge b$ et $s(a, b, 0) = (a \wedge \neg b) \vee (\neg a \wedge b)$.) Le temps de calcul peut être identifié à la profondeur du circuit, c'est-à-dire au plus grand nombre de portes traversées dans un chemin reliant une entrée à une sortie. La cellule de base est de profondeur 5, donc le circuit est de profondeur $5n$ (ou $5n - 2$ en tenant compte de la remarque ci-dessus).

(A3) L'idée est de modifier l'additionneur à un bit pour calculer simultanément la retenue sortant rs et sa négation $\overline{rs}$ (en dédoublant les entrées et les sorties du circuit). On a toujours

$$rs = (a \wedge b) \vee (a \wedge re) \vee (b \wedge re),$$

et

$$\overline{rs} = (\overline{a} \vee \overline{b}) \wedge (\overline{a} \vee \overline{re}) \wedge (\overline{b} \vee \overline{re}).$$

De même on a

$$s = (a \wedge b \wedge re) \vee (a \wedge \overline{b} \wedge \overline{re}) \vee (\overline{a} \wedge b \wedge \overline{re}) \vee (\overline{a} \wedge \overline{b} \wedge re),$$

et

$$\overline{s} = (\overline{a} \vee \overline{b} \vee \overline{re}) \wedge (\overline{a} \vee b \vee re) \wedge (a \vee \overline{b} \vee re) \wedge (a \vee b \vee \overline{re}).$$

On obtient donc un circuit de profondeur $4n +$ Constante (sauf erreur, $4n - 1$ en tenant compte de la remarque de la question précédente). CQFD

(B1) On travaille maintenant avec des circuits dans lesquels les portes $\vee$ et $\wedge$ sont de degré entrant non borné (et bien sûr calculent toujours la disjonction et la conjonction de leurs entrées). Construire un additionneur de profondeur constante, évaluer le nombre de portes. (Indication: supposons qu'une retenue est générée lors du calcul de $C(i)$. A quelle condition cette retenue se propage t-elle jusqu'à la position j?)

(B2) On revient à des portes de degré entrant 2. Adaptez la construction précédente, évaluez la profondeur du circuit et le nombre de portes. Montrez qu'aucun circuit de profondeur inférieure à $\log n$ ne peut calculer l'addition.

(B3) Montrez que toute fonction f de $\{0,1\}^n$ dans $\{0,1\}$ peut être calculée par un circuit de degré entrant non-borné et de profondeur constante.

PREUVE. (B1) La retenue se propage jusqu'à la position j si on a $a(k)+b(k) \geq 1$ pour tous les k de $i+1$ à $j-1$. La retenue entrante en position j s'exprime donc de la façon suivante:

$$r_j = 1 \Leftrightarrow \exists i < j [A(i) \wedge B(i) \wedge (\forall k.\ i < k < j)\ A(k) \vee B(k)] \tag{1}$$

Le circuit correspondant est de profondeur 3. Ensuite on a comme avant $C(n) = r_n$ et $C(i) = s(A(i), B(i), r_i)$ pour $i < n$. Nous obtenons donc un circuit de profondeur $3+5=8$.
Nombre de portes: nous avons besoin pour chaque $k \in \{0, \ldots, n-1\}$ d'une porte $\vee$ pour calculer $A(k) \vee B(k)$, soit n portes. Puis nous avons besoin pour chaque couple (i,j) avec $i < j$ d'une porte $\wedge$ pour calculer la conjonction

$$A(i) \wedge B(i) \wedge (\forall k.\ i < k < j) A(k) \vee B(k),$$

soit $\sum_{j=1}^{n} j = n(n+1)/2$ portes. Pour chaque $j \in \{0, \ldots, n\}$ nous avons besoin d'une porte $\vee$ pour calculer la disjonction "$\exists i < j$", soit $n+1$ portes. Enfin, pour chaque $j \in \{0, \ldots, n-1\}$ nous avons besoin de 11 portes pour calculer $C(j) = s(a(j), b(j), r_j)$, soit $11n$ portes. Au total le nombre de portes est donc équivalent à $n^2/2$.

(B2) On remplace chaque disjonction (ou conjonction) d'arité $N \geq 3$ par un arbre binaire de $N-1$ portes $\vee$ (ou $\wedge$). Cet arbre est de profondeur au plus $1+\log N$. Dans (1) la disjonction est d'arité j et chaque conjonction est d'arité $j-i+1$. Ces arités sont toutes majorées par $n+2$. En reprenant l'analyse de la question 4, on obtient donc un circuit de $O(n^3)$ portes et de profondeur $O(\log n)$.
Borne inférieure: considérons la porte de sortie g_n qui produit le bit $C(n)$. Soit n_k l'ensemble des portes du circuit qui sont à distance k de cette porte. Comme toutes les portes sont d'arité au plus 2, on a $n_{k+1} \leq 2n_k$. Le nombre de portes à distance entre 1 et k de g_n est donc majoré par $\sum_{j=1}^{k} 2^j = 2^{k+1} - 1 < 2^{k+1}$. A fortiori, ceci est une borne supérieure sur le nombre de portes d'entrées à distance au plus k de g_n. Mais on va voir qu'il doit exister un chemin de chacune des $2n$

entrées jusqu'à g_n. Si le circuit est de profondeur d, on doit donc avoir $2^{d+1} \geq 2n$, d'où le résultat.
Montrons par exemple qu'il doit exister un chemin de la porte d'entrée portant le bit $A(i)$ jusqu'à g_n. Si ce n'est pas le cas, c'est que $C(n)$ ne dépend pas de $A(i)$. Mais $C(n)$ dépend bien sûr de $A(i)$, comme on peut le voir en considérant les deux entrées suivantes:

$$A = 0, \ B = 2^n - 2^i,$$

et

$$A = 2^i, \ B = 2^n - 2^i.$$

Dans le premier cas on a $C(n) = 0$, et $C(n) = 1$ dans le second cas. Comme ces deux exemples diffèrent uniquement par la valeur de $A(i)$, c'est bien que $A(i)$ influence $C(n)$.

(B3) On utilise bien sûr la forme normale disjonctive (ou conjonctive), ce qui nous donne un circuit de profondeur 3. L'inconvénient de cette construction est que nous avons une porte $\wedge$ pour chaque entrée x telle que $f(x) = 1$, donc ce circuit peut être de taille exponentielle (mais on peut voir que c'est inévitable). CQFD

2 Calcul d'entiers

On appelle *programme de longueur* k une suite d'instructions $l_0, \ldots, l_k$ de la forme $l_0 : \quad x_0 := 1$ et pour $1 \leq i \leq k$, $l_i : \quad x_i := x_j \perp x_k$ avec $0 \leq i, j < k$ et $\perp \in \{+, -, \times\}$. Le résultat du programme est x_k. Pour $m \in \mathbf{Z}$, on note $\tau(m)$ la longueur du plus petit programme qui calcule m.

(A1) Montrez que $\tau(m) \leq 2 \log m$ pour tout entier $m \geq 1$.

(A2) Existe t-il des entiers de τ beaucoup plus petit ?

(A3) Existe t-il des entiers de τ encore plus petit ?

(A4) Existe t-il des entiers m avec τ proche de $\log m$, donc proche de la borne supérieure de la question 1 ?

(A5) Etant donné deux programmes P_1 et P_2, on souhaite déterminer s'ils calculent le même entier. Décrire un algorithme (sans astuce particulière!) pour résoudre ce problème. Que peut-on dire du temps d'exécution de cet algorithme en fonction de la taille des programmes ?

Preuve. (A1) On observe que $\tau(2m) \leq \tau(m) + 1$ et $\tau(2m+1) \leq \tau(m) + 2$, donc si l'écriture binaire de m est de la forme $b_k \cdots b_0$ on a par récurrence $\tau(m) \leq 2k \leq 2 \log m$ en utilisant la relation $m = 2(b_k \cdots b_1) + b_0$.

(A2) Oui, par exemple si m est de la forme 2^{2^k} on a $\tau(m) \leq k+1 = \log\log m + 1$ par élévations au carré successives puisque $(2^{2^k})^2 = 2^{2^{k+1}}$.

(A3) Non, on montre par récurrence qu'un programme de longueur $k+1$ ne peut pas calculer un entier plus grand que 2^{2^k}.

(A4) On va majorer le nombre de programmes de longueur k. Pour l'instruction l_j on a 3 choix possibles pour l'opération arithmétique, j choix pour l'opérande gauche et j choix pour l'opérande droit, d'où la borne $(3k^2)^k$. On remarque que si un entier est calculé par un programme de longueur $< k$, il peut aussi l'être par un programme de longueur exactement k (il suffit de bourrer le début du programme par des instructions inutiles). Si $3^k k^{2k} < N$, il doit donc exister au moins un entier $m \in \{1, \ldots, N\}$ tel que $\tau(m) > k$. Cette condition est équivalente à

$$k \log 3 + 2k \log k < \log N,$$

et il suffit donc d'avoir $k \log 3 < (\log N)/2$ et $2k \log k < (\log N)/2$. La première de ces inégalités nous donne

$$\log k < \log\log N - 1 - \log\log 3 < \log\log N,$$

donc en reportant dans la second inégalité il suffit d'avoir $k \leq \log N/(4 \log\log N)$, ce qui assure aussi que la première inégalité est satisfaite pour N assez grand. Donc pour N assez grand, en prenant $k = \lfloor \log N/4 \log\log N \rfloor$, il existe $m \in \{1, \ldots, N\}$ tel que $\tau(m) \geq k+1 \geq \log N/4 \log\log N$, ce qui donne

$$\tau(m) \geq \log m/(4 \log\log m) ,$$

toujours pour N assez grand, puisque la fonction $x \mapsto \log x/\log\log x$ est croissante à partir d'un certain rang. (Remarque: ce résultat est assez faible puisqu'il ne garantit l'existence que *d'un seul* m de complexité au moins $m/4 \log\log m$. En fait, il existe une infinité d'entiers m de complexité $\Omega(\log m/\log\log m)$. En effet, les mêmes calculs montrent l'existence d'un entier $m \in [N, 2N[$ de complexité au moins $\log N/4 \log\log N$, ce qui donne cette fois $\tau(m) \geq \log(m/2)/4 \log\log(m/2)$.)

(A5) On peut simplement évaluer les résultats x_1 et x_2 de P_1 et P_2. Si k est un majorant de la taille de P_1 et de la taille de P_2, ceci se fait en au plus $2k$ opérations arithmétiques. Mais les entiers manipulés peuvent avoir jusqu'à 2^{k-1} chiffres binaires, ce qui conduit à un temps d'exécution (et à une place mémoire) exponentielle en k.

(B1) On décide d'effectuer toutes les opérations arithmétiques modulo un entier p. Que peut on dire de la correction d'un tel algorithme ?

(B2) Donner en fonction de k (majorant de la taille des programmes P_1 et P_2) et de N une borne inférieure sur le nombre d'entiers $p \in \{2, \ldots, N\}$ qui conduisent à une bonne réponse. (Indication: si $\pi(x)$ désigne le nombre de nombres premiers inférieurs ou égaux à x, on a $\pi(x) \geq cx/\log x$ pour une certaine constante c.)

(B3) On considère un nouvel algorithme dans lequel on calcule modulo p_i pour plusieurs entiers $p_1, \ldots, p_q$. On déclare que $x_1 = x_2$ si $(x_1 \bmod p_i) = (x_2 \bmod p_i)$ pour tous les p_i, sinon on déclare que $x_1 \neq x_2$. Montrer qu'il existe une constante universelle a telle que pour $q = a.k^2 \ln k$ et k suffisamment grand, il existe q entiers $p_1, \ldots, p_q$ conduisant au bon résultat pour tous les programmes de longueur k, avec tous les p_i majorés par $1 + 2^{2k}$.

(B4) La question (B3) fournit une borne supérieure sur les valeurs des p_i (au plus 2^{2k}) et leur nombre ($ak^2 \ln k$). On cherche maintenant à obtenir une borne inférieure. Supposons donc que les p_i sont tous majorés par $2^{\sqrt{k}}$. Montrez qu'il n'y a pas de bon choix pour $p_1, \ldots, p_q$ si q est trop petit. Quelle borne inférieure sur q obtenez vous ?

PREUVE. (B1) Si $x_1 = x_2$, on donne toujours la bonne réponse quel que soit le choix de p. Mais si $x_1 \neq x_2$, on va se tromper si $x_1 \bmod p = x_2 \bmod p$ (ou autrement dit si p divise $|x_1 - x_2|$). Dans ce cas nous dirons que p est *mauvais* pour le couple de programmes (P_1, P_2).

(B2) $|x_1|$ et $|x_2|$ sont bornés par $2^{2^{k-1}}$, donc nous voulons majorer le nombre de diviseurs d'un entier $\leq 2^{2^k}$. $|x_1 - x_2|$ a au plus 2^k diviseurs premiers, donc dans $\{2, \ldots, N\}$ il y a au moins $cN/\log N - 2^k$ nombres premiers qui ne divisent pas $|x_1 - x_2|$. C'est la borne désirée (c'est une borne inférieure sur le nombre de nombres premiers $\leq N$ qui ne divisent pas $|x_1 - x_2|$, donc a fortiori sur le nombre d'entiers $\leq N$ qui ne divisent pas $|x_1 - x_2|$).

(B3) Supposons qu'on choisit tous les p_i dans $\{2, \ldots, N+1\}$: il y a N^q choix possibles pour la suite $(p_1, \ldots, p_q)$. Pour un couple de programmes fixés, on a vu que le nombre de mauvais entiers est majoré par

$$2^k + (N+1) - c(N+1)/\log(N+1)$$

pour une certaine constante c. Cette borne est elle-même majorée par $2^k + N - c'N/\log N$ pour une autre constante c' (inférieure à c). Pour ce couple de programmes, le nombre de mauvaises suites $(p_1, \ldots, p_q)$ (i.e. telles que tous les p_i sont mauvais pour le couple) est donc majoré par $(2^k + N - c'N/\log N)^q$. Il y a au plus $(3^k k^{2k})^2$ couples de programmes de longueur k, donc au total le nombre de mauvaises suites (i.e. mauvaises pour au moins un programme de longueur k) est majoré par $3^{2k} k^{4k} (2^k + N - c'N/\log N)^q$. Il doit donc exister une bonne suite si

$$3^{2k} k^{4k} (2^k + N - c'N/\log N)^q < N^q. \tag{2}$$

Posons $N = 2^{2k}$ avec k suffisamment grand pour avoir $2^k < c'N/\log N$. En prenant le log des deux membres de (1), comme $\ln(1-x) \leq -x$ il suffit d'avoir

$$q\left(\frac{c'}{\log N} - \frac{2^k}{N}\right) \geq 2k \ln 3 + 4k \ln k,$$

c'est-à-dire en remplaçant N par 2^{2k}:

$$q \geq \frac{2k \ln 3 + 4k \ln k}{\frac{c'}{2k} - \frac{2^k}{2^{2k}}}$$

soit encore en multipliant numérateur et dénominateur par $2k$,

$$q \geq \frac{2k(2k \ln 3 + 4k \ln k)}{c' - k/2^{k-1}},$$

d'où le résultat.

(B4) Supposons que $(p_1, \ldots, p_q)$ est une bonne suite. Soit $m = \prod_{i=1}^{q} p_i$: on a $m = 0 \pmod{p_i}$ pour tous les p_i bien que $m \neq 0$. Mais on a $\tau(0) = 1 \leq k$, et

$$\tau(m) \leq 2 \log m \leq 2 \sum_{i=1}^{q} \log p_i \leq 2q\sqrt{k}.$$

On a donc $\tau(m) \leq k$ pour $q \leq \sqrt{k}/2$, ce qui est la borne cherchée. CQFD

La dernière partie de ce problème est indépendante de ce qui précède, mais illustre de manière peut-être plus limpide la technique utilisée pour le calcul de la borne supérieure de la question (B3).

(C) Soit E un sous-ensemble de $\{0,1\}^n$. On appelle translaté de E, et on note $E+t$, un sous-ensemble de $\{0,1\}^n$ de la forme $\{x+t;\ x \in E\}$ où $t \in \{0,1\}^n$ (les additions sont effectuées modulo 2). On cherche à quelle condition il existe m translatés de E dont l'union est égale à $\{0,1\}^n$. Montrez que $m|E| \geq 2^n$ est une condition nécessaire, et $m|E| > n2^n \ln 2$ une condition suffisante.

Preuve. (C) La condition nécessaire est claire puisque chaque translaté a le cardinal de E. Pour la condition suffisante, on majore le nombre des "mauvaises" suites $(t_1, \ldots, t_m)$ de vecteurs de translation (c'est-à-dire telles que l'union des $E + t_i$ est strictement incluse dans $\{0,1\}^n$). Il y a un total de $(2^n)^m$ suites. Un vecteur de translation t est dit *mauvais* pour un point x si $x \notin E + t$, c'est-à-dire si $t \notin x - E$. Il y a donc $2^n - |E|$ vecteurs qui sont mauvais pour x. Le nombre de mauvaises suites pour x (i.e. telles que chaque t_i est mauvais pour x) est donc $(2^n - |E|)^m$, et le nombre de suites qui sont mauvaises pour au moins un point de $\{0,1\}^n$ est majoré par $2^n(2^n - |E|)^m$. Il doit donc exister une bonne suite si $2^n(2^n - |E|)^m < 2^{mn}$, i.e., $2^n(1 - |E|/2^n)^m < 1$, ou encore

$$-m \ln(1 - |E|/2^n) > n \ln 2.$$

Comme $\ln(1+x) \leq x$, cette condition est bien vérifiée si $m|E| > n2^n \ln 2$. CQFD

3 Théorème de Spira

On considère des termes sur n variables booléennes $x_1, \ldots, x_n$: $x_1, \ldots, x_n$ et les constantes 0 et 1 sont les termes de taille 1; si T_1, T_2 sont des termes de taille respective t_1, t_2, $T_1 \vee T_2$ et $T_1 \wedge T_2$ sont des termes de taille $1 + t_1 + t_2$; $\neg T_1$ est un terme de taille $1 + t_1$.

(A1) Exhibez un terme $S(x, y, z)$ (appelé *selecteur*) vérifiant les propriétés suivantes: $S(0, y, z) = y$ et $S(1, y, z) = z$.

(A2) On souhaite définir une notion de profondeur de terme. On décide que les variables et les constantes sont de profondeur nulle. Comment définir la profondeur pour les autres termes ?

(A3) Montrez que toute fonction de $\{0, 1\}^n$ dans $\{0, 1\}$ peut se représenter par un terme.

(A4) Pour cette question uniquement, on remplace les connecteurs $\vee$, $\wedge$ et $\neg$ par un unique connecteur binaire $\mathrm{NOU}(x, y) = \neg(x \vee y)$. Montrez que les termes construits avec ce connecteur permettent encore d'exprimer toutes les fonctions booléennes.

(A5) On munit $\{0, 1\}$ de l'ordre $0 < 1$, et $\{0, 1\}^n$ de l'ordre induit composante par composante: $x \leq y$ si et seulement $x_i \leq y_i$ pour $i = 1, \ldots, n$. Une fonction $f : \{0, 1\}^n \to \{0, 1\}$ est dite monotone (croissante) si $f(x) \leq f(y)$ chaque fois que $x \leq y$. On dit qu'un terme est monotone s'il ne comprend pas de négations. Montrez qu'une fonction est monotone si et seulement si elle peut-être représentée par un terme monotone.

Preuve. (A1) On peut prendre $S(x, y, z) = (x \wedge z) \vee (\neg x \wedge y)$.

(A2) Comme pour la taille, on raisonne par induction sur la structure des termes: la profondeur de $T_1 \vee T_2$ et $T_1 \wedge T_2$ est $1 + \max(\text{profondeur}(T_1), \text{profondeur}(T_2))$. La profondeur de $\neg T$ est $1 + \text{profondeur}(T)$.

(A3) On peut utiliser la forme normale disjonctive, ou la forme normale conjonctive, ou avec un oeil sur la question (A5) raisonner par récurrence sur le nombre de variables en écrivant:

$$f(x_1, \ldots, x_n) = (f(x_1, \ldots, x_{n-1}, 0) \wedge \neg x_n) \vee (f(x_1, \ldots, x_{n-1}, 1) \wedge x_n).$$

Etant donné des termes pour les fonctions de $n-1$ variables $(x_1,\ldots,x_{n-1}) \mapsto f(x_1,\ldots,x_{n-1},0)$ et $(x_1,\ldots,x_{n-1}) \mapsto f(x_1,\ldots,x_{n-1},1)$, cette écriture nous donne immédiatement un terme pour f.

(A4) Il suffit de montrer qu'on peut construire les connecteurs $\neg$, $\vee$ et $\wedge$ à partir de NOU. On a $\neg x =$ NOU (x,x), puis $x \vee y = \neg$ NOU (x,y) et $x \wedge y =$ NOU $(\neg x, \neg y)$.

(A5) On montre facilement par induction sur la structure des termes que tout terme monotone calcule une fonction monotone (utiliser la monotonie de $\vee$ et $\wedge$). La réciproque est bien vraie pour $n=1$ (les seules fonctions monotones sont les constantes et l'identité). Pour $n>1$, on écrit cette fois:

$$f(x_1,\ldots,x_n) = f(x_1,\ldots,x_{n-1},0) \vee (f(x_1,\ldots,x_{n-1},1) \wedge x_n).$$

En effet, si $f(x_1,\ldots,x_{n-1},0)=1$ alors $f(x_1,\ldots,x_{n-1},1)=1$ par monotonie de f. Comme les fonctions $(x_1,\ldots,x_{n-1}) \mapsto f(x_1,\ldots,x_{n-1},0)$ et $(x_1,\ldots,x_{n-1}) \mapsto f(x_1,\ldots,x_{n-1},1)$ sont monotones, on conclut avec l'hypothèse de récurrence. CQFD

(B1) Soit T un terme de taille t sur les variables $x_1,\ldots,x_n$. On suppose que x_n est un sous-terme strict de T et n'apparaît qu'une seule fois dans T. Montrez que $T(x_1,\ldots,x_{n-1},0)$ et $T(x_1,\ldots,x_{n-1},1)$ sont équivalents à des termes de taille $\leq t-1$.

(B2) Montrez que tout terme Θ de taille t est équivalent à un terme de profondeur $\leq 4\log t$. (Indication: on considérera un sous-terme T de Θ minimal de taille strictement supérieure à $t/2$.)

PREUVE. (B1) On le fait par exemple pour $T(x_1,\ldots,x_{n-1},0)$. Puisque x_n est un sous-terme strict, il existe un sous-terme T' de T dont x_n est un sous-terme immédiat. On considère les trois cas possibles. Si $T' = \neg x_n$ on peut remplacer T' par la constante 1, et la taille du terme diminue bien. Si T' est de la forme $x_n \wedge U$ on peut remplacer T' par la constante 0. Enfin, si T' est de la forme $x_n \vee U$, on peut remplacer T' par U.

(B2) On raisonne par récurrence sur t. Le résultat est vrai pour $t=1$ puisque les termes de taille 1 sont de profondeur nulle. Considérons maintenant un terme Θ de taille $t \geq 2$, et soit T choisi selon l'indication. Comme T n'est pas de taille 1, il a un ou deux sous-termes immédiats, qui par minimalité de T sont de taille au plus $t/2$, donc strictement inférieure à t. Par hypothèse d'induction, T est donc équivalent à un terme T^* de profondeur $1+4\log(t/2) = 4\log t - 3$. Si $T = \Theta$, c'est terminé. Sinon, soient U_0 et U_1 les termes obtenus à partir de Θ en remplaçant T par la constante 0 et la constante 1, respectivement. Ces deux termes sont de taille au plus $t/2 + 1$, et d'après la question précédente ils sont équivalents à des termes de taille au plus $t/2$. Par hypothèse d'induction, ces termes sont équivalents à des termes de profondeur $4\log(t/2) = 4\log t - 4$. Finalement, comme Θ est équivalent à $S(T, U_0, U_1)$ et que le sélecteur se calcule en profondeur 3, Θ est équivalent à un terme de profondeur au plus $3 + (4\log t - 3) = 4\log t$. CQFD

4 Couplage maximum

Source. "Introduction to Graph Theory" de D. West.

Soit $G = (X, Y, E)$ un graphe biparti construit sur deux ensembles indépendants X et Y (toute arête $e \in E$ relie donc un sommet de X à un sommet de Y, et $X \cap Y = \emptyset$). Un couplage est un sous-ensemble $M \subseteq E$ tel que chaque sommet de G est une extrémité d'une arête de M au plus.

(A1) On dit que M est un couplage maximal s'il n'est inclus strictement dans aucun autre couplage. Donner un algorithme de construction d'un couplage maximal.

(A2) On dit que M est un couplage maximum s'il n'existe pas de couplage de cardinalité strictement supérieure. Tout couplage maximal est il maximum?

(A3) Un chemin est une suite de sommets distincts telle que deux sommets consécutifs sont reliés par une arête. Soit M un couplage. Un chemin M-alterné est un chemin qui alterne entre arêtes de M et arêtes de $E \setminus M$, et un chemin M-augmentant est un chemin M-alterné qui commence et finit par des sommets M-insaturés, c'est-à-dire des sommets qui ne sont extrémité d'aucune arête de M. Montrez que s'il existe un chemin M-augmentant alors M n'est pas maximum.

(A4) On appelle *couverture* un ensemble C de sommets de G tel que toute arête du graphe a au moins l'une de ses deux extrémités dans C. Soit C une couverture et M un couplage. Comparez $|C|$ et $|M|$.

Preuve. (A1) L'algorithme peut être spécifié comme suit:

$M = \emptyset$
Tant qu'il existe une arête e de $E \setminus M$ ne partageant aucune extrémité avec une arête de M,
    ajouter e à M.
Retourner M.

(A2) Non, et le graphe suivant donne un contre-exemple: $X = \{a, b\}$, $Y = \{1, 2\}$, les arêtes sont $a1$, $b1$ et $b2$. $\{b1\}$ est un couplage maximal non maximum.

(A3) Soit N l'ensemble des arêtes de M empruntées par le chemin augmentant, et O l'ensemble des autres arêtes (de $E \setminus M$) empruntées par ce chemin. $O \cup (M \setminus N)$ est un couplage de cardinalité $|M| + 1$.

(A4) On a $|C| \geq |M|$ puisque les arêtes de M doivent être couvertes par des sommets distincts. CQFD

(B1) On considère l'algorithme suivant, qui construit deux ensembles de sommets $S \subseteq X$ et $T \subseteq Y$. U dénote l'ensemble des sommets M-insaturés de X. Au départ tous les sommets de U sont non marqués.

$S = U$
$T = \emptyset$
tant qu'il y a dans S des sommets non marqués faire:
    Choisir un sommet $x \in S$ non marqué.
    Pour chaque voisin y de x tel que $y \notin T$ et $xy \notin M$ faire:
        Si y est insaturé, arrêter en répondant "M n'est pas maximum".
        Sinon, soit $w \in X$ tel que $yw \in M$.
        Ajouter y à T.
        Ajouter w à S.
    Marquer x.
Arrêter en répondant "$T \cup (X \setminus S)$ est une couverture".

Montrez que l'algorithme termine toujours, et qu'il est correct.

(B2) Dans le cas où l'algorithme retourne une couverture, comparez la taille de cette couverture à celle de M. Qu'en déduisez vous sur M?

(B3) Comment modifier l'algorithme pour construire un couplage maximum ?

(B4) On a vu que s'il existe un chemin M-augmentant, M n'est pas maximum. Que dire de la réciproque ?

PREUVE. (B1) Terminaison: à chaque exécution complète de la boucle "tant que" on marque un sommet de X non marqué, donc cette boucle ne peut être exécutée plus de $|X|$ fois. Par construction, pour tout sommet $x \in S$ il existe un chemin $\gamma(x)$ M-alterné d'un sommet de U à x (réduit à x si $x \in U$). Ceci se montre par récurrence: quand on ajoute w à S on peut prendre $\gamma(w) = \gamma(x)yw$. Il faut vérifier que y et w n'apparaissent pas dans $\gamma(x)$. C'est vrai pour y à cause de la condition $y \notin T$. C'est vrai pour w car avant d'ajouter w à S on a $w \notin S$. En effet, on ne peut pas avoir $w \in U$ puisque les sommets de U sont M-insaturés, et on ne peut pas non plus avoir $w \in S \setminus U$ puisque les sommets de $S \setminus U$ sont reliés par une arête de M à un sommet de T (et donc comme M est un couplage ils ne

peuvent être reliés également à un sommet de $Y \setminus T$). Dans le cas où l'algorithme s'arrête en répondant "M n'est pas maximum", en prolongeant $\gamma(x)$ jusqu'à y on obtient un chemin M-augmentant, donc effectivement M n'est pas maximum.
Il reste donc à montrer que si l'algorithme s'arrête en retournant une couverture, $T \cup (X \setminus S)$ est bien une couverture, c'est-à-dire qu'il n'existe pas d'arête entre S et $Y \setminus T$. Tout sommet x de $S \setminus U$ est relié à un sommet de T par une arête de M (en fait par la dernière arête de $\gamma(x)$), donc, par définition d'un couplage, x ne peut pas être relié également par une arête de M à un sommet de $Y \setminus T$. En fait, aucun sommet de S n'est relié par une arête de M à un sommet de $Y \setminus T$ puisque les sommets de U sont M-insaturés. Un sommet x de S ne peut pas non plus être relié à $Y \setminus T$ par une arête de $E \setminus M$ puisque dans la boucle Pour, tout sommet y qui vérifierait $y \notin T$ et $xy \in E \setminus M$ est ajouté à T. Donc effectivement x n'est relié à aucun sommet de $Y \setminus T$.

(B2) Tout sommet de $X \setminus S$ est une extrémité d'une arête de M puisque l'ensemble U des sommets M-insaturés de X est inclus dans S. De même en utilisant les notations de l'algorithme, tout sommet y de T est relié par une arête de M à un sommet w de S. Toutes les arêtes de M ainsi obtenues sont distinctes donc $|M| \geq |T| + |X \setminus S|$. Mais on a vu que $|M| \leq |T| + |X \setminus S|$, donc en fait $|M| = |T| + |X \setminus S|$. Comme aucun couplage ne peut être plus grand que la couverture $T \cup (X \setminus S)$, M est un couplage maximum.

(B3) Si pour chaque sommet $x \in T$ on se souvient du chemin $\gamma(x)$, on pourra construire un chemin M-augmentant dans le cas où M n'est pas maximum, et donc on pourra retourner un "couplage augmenté" de taille supérieure. Cela donne:

```
S = U
T = ∅
tant qu'il y a dans S des sommets non marqués faire:
        Choisir un sommet x ∈ S non marqué.
        Pour chaque voisin y de x tel que y∉T et xy∉M faire:
                Si y est insaturé, arrêter en retournant un couplage augmenté
                Sinon, soit w ∈ X tel que yw ∈ M.
                Ajouter y à T; precedent(y)=x
                Ajouter w à S; precedent(w)=y
        Marquer x.
Arrêter en répondant "T ∪ (X \ S) est une couverture minimum
et M un couplage maximum".
```

Le couplage augmenté se construit à partir d'un chemin M-augmentant grâce à la réponse à la question 3, et celui s'obtient grâce au tableau des précédents. Il suffit d'appliquer itérativement cet algorithme (en partant par exemple du couplage vide) pour construire des couplages de plus en plus grands, jusqu'à atteindre un couplage maximum. Le nombre d'itérations est majoré par $\min(|X|, |Y|)$.

On peut donner une description plus formelle de l'algorithme, à l'aide de règles d'inférence. On reformule d'abord l'algorithme de la question (B1). On manipule des triplets (S, S', T) tels que $S' \subseteq S \subseteq X$ et $T \subseteq Y$ (S' est l'ensemble des sommets marqués). Le triplet initial est $(U, \emptyset, \emptyset)$. Il y a trois règles d'inférence pour la réécriture de triplets. On applique la seconde seulement si la première ne s'applique pas. La notation $Y \setminus M$ désigne le sous-ensemble de Y des sommets qui ne sont pas extrémité d'une arête de M.

$$\frac{x \in S \setminus S' \qquad \exists y\ \{x,y\} \in E \setminus M, y \notin T \text{ et } y \in Y \setminus M}{(S, S', T) \ \rightarrow\ \text{“}M \text{ n'est pas maximum”}}$$

$$\frac{\begin{array}{c} x \in S \setminus S' \quad \{y_1, \ldots, y_n\} = \{y \mid \{x,y\} \in E \setminus M \text{ et } y \notin T\} \\ \{y_1, w_1\} \in M, \ldots, \{y_n, w_n\} \in M \quad (n \geq 0) \end{array}}{(S, S', T) \ \rightarrow\ (S \cup \{w_1, \ldots, w_n\}, S' \cup \{x\}, T \cup \{y_1, \ldots y_n\})}$$

$$\frac{S \setminus S' = \emptyset}{(S, S', T) \ \rightarrow\ \text{“}T \cup (X \setminus S) \text{ est une couverture”}}$$

Par exemple, sur le graphe donné par

$$\begin{array}{rcl} X & = & \{a, b, c, d, e, f, g, h, i\} \\ Y & = & \{d', e', f', g', h', i', j', k'\} \\ E & = & \{(a, d'), (b, d'), (d, d'), (e, d'), (c, e'), (e, e'), (f, e'), (h, e'), (i, e'), \\ & & (e, f'), (f, f'), (g, f'), (g, g'), (h, h'), (i, i'), (h, j'), (i, k')\} \end{array}$$

et en prenant $M = \{(d, d'), (e, e'), (f, f'), (g, g'), (h, h'), (i, i')\}$ l'algorithme se déroule comme suit

$$\begin{array}{rcl} (\{a, b, c\}, \emptyset, \emptyset) & \rightarrow & (\{a, b, c, d\}, \{a\}, \{d'\}) \\ & \rightarrow & (\{a, b, c, d\}, \{a, b\}, \{d'\}) \\ & \rightarrow & (\{a, b, c, d, e\}, \{a, b, c\}, \{d', e'\}) \\ & \rightarrow & (\{a, b, c, d, e\}, \{a, b, c, d\}, \{d', e'\}) \\ & \rightarrow & (\{a, b, c, d, e, f\}, \{a, b, c, d, e\}, \{d', e', f'\}) \\ & \rightarrow & (\{a, b, c, d, e, f\}, \{a, b, c, d, e, f\}, \{d', e', f'\}) \\ & \rightarrow & \text{“}\{d', e', f'\} \cup \{g, h, i\} \text{ est une couverture”} \end{array}$$

Pour formaliser l'algorithme modifié, on transporte M comme quatrième composante dans le système formel, et on remplace les deux premières composantes par des ensembles de chemins. Les ensembles de sommets sont vus comme des ensembles de chemins (de longueur 0). On désigne par $b(\gamma)$ le sommet final du chemin γ. On note M_γ^+ le couplage construit en (A3) (γ étant le chemin M-augmentant). On commence avec le quadruplet initial $(X, \emptyset, \emptyset, \emptyset)$.

$$\frac{\gamma \in S \setminus S' \qquad \exists\, y \;\; \{b(\gamma), y\} \in E \setminus M, y \notin T \text{ et } y \in Y \setminus M}{(S, S', T, M) \;\to\; (X - M_{\gamma y}^+, \emptyset, \emptyset, M_{\gamma y}^+)}$$

$$\frac{\begin{array}{c}\gamma \in S \setminus S' \quad \{y_1, \ldots, y_n\} = \{y \mid \{b(\gamma), y\} \in E \setminus M \text{ et } y \notin T\} \\ \{y_1, w_1\} \in M, \ldots, \{y_n, w_n\} \in M \quad (n \geq 0)\end{array}}{(S, S', T, M) \;\to\; (S \cup \{\gamma y_1 w_1, \ldots, \gamma y_n w_n\}, S' \cup \{\gamma\}, T \cup \{y_1, \ldots y_n\}, M)}$$

$$\frac{S \setminus S' = \emptyset}{(S, S', T, M) \;\to\; \text{“}M \text{ est maximum”}}$$

La preuve de terminaison est immédiate: ordre lexicographique sur la quatrième composante, puis sur la deuxième.

(B4) Si M n'est pas maximum, l'algorithme de la question précédente construit un chemin augmentant, donc la réciproque est vraie également. Il y a aussi une preuve directe ([West, théorème 3.1.6] ou Berge, 1957). CQFD

5 Mariages stables

Source. D'après "Introduction to Graph Theory" de D. West pour les questions 1 à 4.

Le problème de l'internat stable: Il y a $2n$ internes et on doit mettre deux internes par chambre. Chaque interne a une liste de préférences qui ordonne totalement l'ensemble des $2n-1$ autres internes. On dit que l'internat est instable s'il existe deux internes a et b (appelés *éléments perturbateurs*) tels que a préfère b à son camarade de chambre, et b préfère a à son camarade de chambre.

(A) Montrez que dans le cas suivant il n'y a pas d'internat stable:

$$\begin{array}{l} a : b > c > d \\ b : c > a > d \\ c : a > b > d \\ d : a > b > c \end{array}$$

Preuve. (A) Il n'y a que trois internats possibles.

- $(a, b), (c, d)$ instable: b, c sont perturbateurs.
- $(a, c), (b, d)$ instable: a, b sont perturbateurs.
- $(a, d), (b, c)$ instable: a, c sont perturbateurs. CQFD

Dans la suite du problème on cherche à marier n hommes à n femmes. Un mariage est donc une bijection de l'ensemble des hommes dans l'ensemble des femmes. Chaque homme a une liste de préférences qui ordonne totalement l'ensemble des femmes, et réciproquement. On dit qu'un mariage est stable s'il n'existe pas une femme f et un homme h' tels que f préfère h' à son conjoint, et réciproquement h' préfère f à son conjoint (autrement dit, on veut éviter tout risque d'adultère).

(B1) On considère l'algorithme de mariage suivant.

> Répéter
>
> Chaque homme demande en mariage la femme la plus haute sur sa liste qui ne l'a pas déjà rejeté.
>
> Toute femme qui reçoit $k \geq 2$ demandes en mariage rejette les demandes émanant des $k-1$ hommes les plus bas sur sa liste.
>
> Jusqu'à ce que chaque femme reçoive exactement une demande.
>
> On marie alors chaque femme à l'homme qui l'a demandée.

Quel est le mariage produit par l'algorithme sur l'exemple suivant?

$$\begin{array}{ll} Hommes : \{x, y, z, w\} & Femmes : \{a, b, c, d\} \\ x : a > b > c > d & \quad a : z > x > y > w \\ y : a > c > b > d & \quad b : y > w > x > z \\ z : c > d > a > b & \quad c : w > x > y > z \\ w : c > b > a > d & \quad d : x > y > z > w \end{array}$$

(B2) Montrez que cet algorithme termine toujours et produit un mariage stable.

(B3) Montrez que cet algorithme produit toujours un mariage M qui est optimal du point de vue des hommes. C'est-à-dire que si M' est un autre mariage stable, aucun homme n'est marié dans M' à une femme qu'il préfère à son épouse dans M. Réciproquement, montrez que ce mariage stable est le plus mauvais du point de vue des femmes.

PREUVE. (B1)

- itération 1: a rejette y, c rejette z.
- itération 2: c rejette y.
- itération 3: x épouse a, y épouse b, z épouse d, w épouse c.

(B2) On remarque qu'un homme qui a demandé une femme en mariage continue à demander la même femme à toutes les itérations ultérieures jusqu'à être rejeté, ou marié à la fin de l'algorithme. Toute femme demandée en mariage à une certaine itération est donc demandée également à toutes les itérations ultérieures (éventuellement par des hommes différents). Ceci permet de montrer qu'à chaque étape tous les hommes font une demande (a priori on pourrait imaginer qu'un homme soit rejeté par toutes les femmes, et il ne pourrait plus faire de demandes aux itérations ultérieures). En effet, supposons par l'absurde qu'un homme h est rejeté par toutes les femmes. Aux itérations ultérieures toutes les femmes reçoivent une demande puisqu'elles ont un jour été demandées par h. C'est impossible sinon il y aurait plus d'hommes que de femmes.
Terminaison: au départ la somme des longueurs des listes de chaque homme est n^2. Si à la fin d'une itération la condition d'arrêt n'est pas vérifiée c'est qu'au moins une femme est demandée par plusieurs hommes (car comme on l'a vu, à chaque étape tous les hommes font une demande). Dans ce cas au moins un

homme est rejeté, et la somme des longueurs des listes des femmes demandables diminue d'autant. Il ne peut donc y avoir plus de n^2 itérations. L'algorithme termine donc toujours, et produit un mariage.
Stabilité: Supposons que f et h' sont tels que h' préfère f à son conjoint. C'est donc que f a rejeté h'. Et comme f préfère son conjoint à tous ceux qu'elle a rejetés, alors f préfère en particulier son conjoint à h', et il n'y a donc pas de danger d'adultère.

On peut donner une version plus formelle de l'algorithme, ainsi que des arguments de terminaison et de correction. On fait évoluer un tableau L indexé par les hommes, dont chaque élément est une liste de femmes. Le tableau est initialisé avec les listes de préférence complètes de chaque homme. Puis on réécrit L en L', notation $L \to L'$, avec L' défini comme suit:

$$L'(h) = \begin{cases} l \text{ si } L(h) = f :: l \text{ et } \exists\, h' \ \ (L(h') = f :: l' \text{ et } f \text{ préfère } h' \text{ à } h) \\ L(h) \text{ sinon .} \end{cases}$$

où f (l) est la tête (queue) de $L(h)$. La terminaison est évidente: la taille de L (somme des longueurs des listes $L(h)$) diminue à chaque étape. On définit une fonction partielle $\mathit{Demande}(L)$ des hommes vers les femmes comme suit: $\mathit{Demande}(L)$ est définie si et seulement si $L(h)$ est non vide, et dans ce cas est égal à la première femme de la liste $L(h)$. Si $L \to L'$, on a $\mathit{Image}(\mathit{Demande}(L)) \subseteq \mathit{Image}(\mathit{Demande}(L'))$. Cet invariant nous permet de montrer que si $L \to^\star L'$, alors $L'(h)$ n'est vide pour aucun homme h. Supposons que $L'(h)$ est vide. Alors chaque femme f a été demandée par h à une étape L'' intermédiaire entre L et L'. Comme $f \in \mathit{Image}(\mathit{Demande}(L''))$, on a $f \in \mathit{Image}(\mathit{Demande}(L'))$. Donc $\mathit{Image}(\mathit{Demande}(L'))$ a le cardinal n (le nombre total de femmes), et a fortiori le domaine de définition de $\mathit{Demande}(L')$ est de cardinal $\geq n$. Comme h n'est pas dans le domaine de définition de $\mathit{Demande}(L')$, c'est qu'il y a strictement plus d'hommes que de femmes, contrairement à l'hypothèse. Donc $\mathit{Demande}(L')$ est totale. Si de plus L' est un état final, alors $\mathit{Demande}(L')$ est injective: donc tout homme est marié à une femme différente à la fin de l'algorithme.

(B3) Supposons que M n'est pas optimal. C'est qu'il existe un mariage stable M' et un couple (h, f') marié dans M' tel que f' a rejeté h dans l'algorithme. Si f' a rejeté h, c'est qu'elle a été demandée en même temps par un homme h' qu'elle préfère à h. Soit f'' la femme de de h' dans M'. Par stabilité de M', h' préfère son épouse f'' à f'.
A partir du couple (h, f') nous avons donc construit un couple (h', f'') vérifiant les mêmes propriétés: h' est marié à f'' dans M' mais f'' a rejeté h' dans l'algorithme

(puisque dans M, h' se retrouve au mieux marié à f'). De plus, le rejet de h' par f'' a eu lieu a une itération antérieure au rejet de h par f'. En effet h' ne demande f' en mariage qu'après avoir été rejeté par f''. On arrive donc à une contradiction en choisissant le couple (h, f') de sorte que le rejet de h par f' ait lieu le plus tôt possible pendant le déroulement de l'algorithme.
La propriété de pessimalité du point de vue des femmes découle de celle d'optimalité du point de vue des hommes. Supposons en effet que f est mariée à h dans M, et est mariée dans un autre mariage stable M' a un homme h' qu'elle aime moins que h. Dans M', h est marié à une femme f' qu'il préfère à f par stabilité de M'. Mais M ne serait alors pas optimal puisque h y est marié à f.
Naturellement, on peut obtenir un mariage stable optimal pour les femmes, et pessimal pour les hommes, avec l'algorithme inversé dans lequel ce sont les femmes qui font les propositions. CQFD

(C1) On ne suppose plus qu'il y a autant d'hommes que de femmes. Définir une notion de stabilité (on supposera que tout le monde préfère être marié plutôt que célibataire). Proposez un algorithme de mariage. Propriétés de cet algorithme.

(C2) Avec les hypothèses de la question précédente, montrez que l'ensemble des personnes qui restent célibataires est le même dans tous les mariages stables.

(C3) Si une personne reste célibataire dans un mariage stable, aurait-il pu en aller autrement si elle avait une liste de préférences différente?

Preuve. (C1) Notion de stabilité: c'est essentiellement la même. On dit que le mariage est stable s'il n'existe pas une femme f et un homme h tels que h préfère f à sa situation actuelle (célibataire ou marié à une femme moins bien classée que f) et réciproquement f préfère h à sa situation actuelle. L'algorithme aussi est très semblable:

Répéter
    Chaque homme demande en mariage la femme la plus haute sur sa
liste qui ne l'a pas déjà rejeté.
    Toute femme qui reçoit $k \geq 2$ demandes en mariage rejette les
        demandes émanant des $k-1$ hommes les plus bas sur sa liste.
Jusqu'à ce qu'aucune femme ne reçoive plus d'une demande.
On marie alors chaque femme demandée à l'homme qui l'a demandée.

Propriétés de l'algorithme: soit m le nombre d'hommes et n le nombre de femmes. Le raisonnement de la question (B1) montre que l'algorithme termine en au plus mn itérations. Premier cas: $m < n$. Comme tous les hommes font une demande à toutes les itérations, l'algorithme va marier tous les hommes, mais des femmes resteront célibataires. Dans le deuxième cas ($m > n$) des hommes devront rester célibataires. Mais un homme qui reste célibataire a fait des propositions a toutes les femmes (et a été rejeté par toutes). Comme toute femme qui reçoit un jour une proposition finit mariée, l'algorithme marie toutes les femmes. Les mêmes raisonnements montrent que l'algorithme produit un mariage stable, et optimal du point de vue des hommes (par stabilité de M', f'' existe toujours).

(C2) Soit M le mariage produit par l'algorithme et M' un autre mariage stable. Supposons d'abord que $m > n$. Par optimalité de M, tous les hommes mariés dans M' sont mariés dans M. Par stabilité, le nombre d'hommes mariés est le même dans M et M' (il est égal à n). Ce sont donc les mêmes hommes qui sont mariés dans M ou M'. Si $m < n$, il suffit de faire le même raisonnement avec l'algorithme inversé dans lequel ce sont les femmes qui font les propositions.

(C3) D'après la question précédente, on peut supposer que les mariages sont produits par l'algorithme de la question 5, et que la personne est une femme (si c'est un homme, on raisonne avec l'algorithme inverse dans lequel ce sont les femmes qui font les propositions). Avec cet algorithme, une femme est mariée si et seulement si elle est un jour demandée en mariage. Si une femme n'est pas demandée en mariage, sa liste de préférences n'intervient pas dans le déroulement de l'algorithme et il ne sert donc à rien de la modifier. CQFD

6 Mariage randomisé

Source. D'après "Randomized Algorithms", Motwani - Raghavan, Cambridge University Press [MR].

Soit $G = (U, V, E)$ un graphe biparti construit sur deux ensembles disjoints $U = \{u_1, \ldots, u_n\}$ et $V = \{v_1, \ldots, v_n\}$ de même cardinalité (l'ensemble E des arêtes est donc inclus dans $U \times V$, et $U \cap V = \emptyset$). Un *mariage parfait* M est un sous-ensemble de E tel que chaque sommet de G est une extrémité d'une arête de M et d'une seule. A G on associe une matrice $n \times n$ construite de la façon suivante: l'entrée A_{ij} est égal à 0 si $(u_i, v_j) \notin E$, et à la variable x_{ij} sinon. Le déterminant de A est donc un polynôme en les variables x_{ij}.

(A1) Montrez que G a un mariage parfait si et seulement si $\det(A)$ n'est pas le polynôme identiquement nul.

(A2) Peut-on en déduire un algorithme pour décider si G a un mariage parfait ? Que dire de sa complexité ?

Remarque: La question (A1) peut se généraliser à un graphe quelquonque (non nécessairement biparti). C'est le théorème de Tutte (exercice 7.8 de [MR]).

PREUVE. (A1) On peut identifier un mariage parfait à une permutation σ de $[n] = \{1, \ldots, n\}$ envoyant tout $i \in [n]$ sur un $j \in [n]$ tel que $(u_i, v_j) \in E$. On a donc:

$$\det(A) = \sum_{\sigma \in S_n} \epsilon(\sigma) A_{1\sigma(1)} \cdots A_{n\sigma(n)} = \sum_{\sigma \text{ mariage parfait}} \epsilon(\sigma) x_{1\sigma(1)} \cdots x_{n\sigma(n)}.$$

Si G n'admet pas de mariage parfait, il ne reste plus de terme dans cette somme, donc $\det(A) \equiv 0$. Réciproquement, tous les monômes dans cette somme étant distincts, si G admet un mariage parfait alors le polynôme n'est pas identiquement nul.

(A2) Une solution naïve consiste à développer formellement le déterminant par une méthode quelquonque d'algèbre linéaire (développement suivant une ligne ou une colonne, Gauss...). Mais toute solution de ce type est nécessairement exponentielle dans le pire des cas puisque le polynôme à calculer peut avoir un nombre exponentiel de monômes. Par exemple, pour un graphe biparti complet, il y a $n!$ mariages parfaits. CQFD

(B1) On décide maintenant plutôt d'évaluer $\det(A)$ en un point choisi au hasard dans $[m]^{n^2}$, où $[m]$ désigne l'ensemble des entiers compris entre 1 et m (on verra plus tard comment choisir m). Que peut-on dire de la validité de l'algorithme correspondant ?

(B2) Soit P un polynôme de degré d en s variables à coefficients entiers. Montrez que si P n'est pas identiquement nul la proportion des points $x \in [m]^s$ tels que $P(x) = 0$ est majorée par d/m (on remarque que s n'intervient pas dans cette borne). Cette borne est-elle optimale ? Application à l'algorithme de la question précédente.

(B3) Quel est l'avantage de l'algorithme obtenu sur l'algorithme naïf ? Que peut on dire de sa complexité ?

PREUVE. (B1) L'algorithme consiste donc à évaluer $\det(A)$ en un point $x \in [m]^{n^2}$ et à répondre:

- "il n'existe pas de mariage parfait" si $(\det A)(x) = 0$,
- "il existe un mariage parfait" si $(\det A)(x) \neq 0$.

S'il n'y a pas de mariage parfait, l'algorithme donne toujours la bonne réponse puisque $\det(A)$ est identiquement nul. Mais s'il en existe un, l'algorithme peut se tromper si on a la malchance de tomber sur un point x qui est une racine de $\det(A)$.

(B2) La borne est optimale dès que $m \geq d$: prendre

$$P(x_1, \ldots, x_s) = (x_1 - 1)(x_1 - 2) \cdots (x_1 - d) .$$

Application: la probabilité de se tromper (pour la distribution uniforme sur $[m]^{n^2}$) est bornée par n/m et on peut la rendre aussi petite qu'on veut en prenant m suffisamment grand.
On pourrait aussi envisager un algorithme déterministe qui évalue P en tous les points de $[m]^s$. D'après la borne, dès que $m \geq d+1$ ce polynôme ne peut s'annuler sur tous les points de l'ensemble, sauf s'il est identiquement nul. Dans notre cas $d = n$ et $s = n^2$, donc on devrait évaluer P en $(n+1)^{n^2}$ points. Ce n'est pas intéressant.
Preuve de la borne: par récurrence sur s. C'est clair pour $s = 1$ (un polynôme de degré d a au plus d racines). Supposons maintenant que $s > 1$, $d \geq 1$ et que

la variable x_1 (par exemple) apparaît dans P. On peut écrire:

$$P(x_1, \ldots, x_s) = \sum_{i=0}^{k} x_1^i Q_i(x_2, \ldots, x_s)$$

avec $1 \leq k \leq d$ et Q_k non identiquement nul (remarquons que ce polynôme est de degré $d - k$). Soit $Z(P)$ l'ensemble des zéros de P, et soit $x \in Z(P)$. On distingue deux cas:

- $Q_k(x_2, \ldots, x_s) \neq 0$, mais le polynôme univarié $P(., x_2, \ldots, x_s)$ s'annule en x_1: d'après le cas de base de la récurrence, il y a au plus $m^{s-1} \times k$ points x dans ce cas.

- $Q_k(x_2, \ldots, x_s) = 0$: par hypothèse de récurrence, il y a au plus $(m^{s-1}(d - k)/m) \times m$ points x dans ce cas.

Le cardinal de $Z(P)$ est donc majoré par $m^{s-1} \times k + m^{s-1}(d - k) = dm^{s-1} = (d/m)m^s$, ce qui donne la borne.

(B3) L'avantage est que $(\det A)(x)$ peut-être évalué en un nombre polynomial d'opérations arithmétiques (par exemple $O(n^3)$ opérations avec la méthode de Gauss). Bien sûr ne surtout pas développer par rapport à une ligne ou à une colonne...

Remarques: La complexité en bits du calcul du déterminant est polynomiale également (voir par exemple Knuth, tome 2) – ceci est non trivial, car a priori on pourrait craindre une explosion de la taille des coefficients de la matrice avec la méthode de Gauss (voir le sujet sur les calculs d'entiers).
Il existe aussi une méthode générale pour éviter l'explosion des coefficients. Au lieu de considérer $P = \det A$ comme un polynôme à coefficients dans $\mathbf{Z}$, on le considère comme un polynôme à coefficients dans le corps $F_p = \mathbf{Z}/p\mathbf{Z}$ avec p un nombre premier. On prend $p \geq \max(m, M + 1)$, avec M la valeur absolue du plus petit coefficient non nul de P (dans notre application, on a $M = 1$ quand il existe un mariage parfait). Si P vu comme polynôme à coefficients dans $\mathbf{Z}$ n'est pas identiquement nul, la condition $p \geq M + 1$ assure qu'il n'est pas non plus identiquement nul en tant que polynôme à coefficients dans F_p. La question (B2) fonctionne alors dans F_p sans modifications: la proportion des points $x \in [m]^s \subseteq (F_p)^s$ tels que $P(x) = 0$ est majorée par d/m. On évite l'explosion des coefficients puisque, dans le calcul de $\det A(x)$ par la méthode de Gauss, tous les calculs se font dans F_p.

(C1) On joue à pile ou face avec une pièce truquée, c'est-à-dire qu'à chaque tirage la probabilité d'obtenir pile est égale à un paramètre $p \in]0,1[$ fixé (on suppose que les tirages sont indépendants). Combien de tirages faut-il effectuer en moyenne pour obtenir le premier pile? Application: On évalue un polynôme P de s variables non identiquement nul en des points aléatoires de $[m]^s$. Combien de tirages faut-il pour en moyenne pour tirer un point où P ne s'annule pas?

(C2) Comment utiliser l'algorithme de la question (B1) pour effectivement construire un mariage parfait ?

PREUVE. (C1) Soit T la variable aléatoire "instant du premier pile". Par définition, la valeur moyenne $E(T)$ (ou "espérance") de T est égale à

$$\sum_{t=0}^{\infty} t \Pr(T = t) = \sum_{t=1}^{+\infty} \Pr(T \geq t) = \sum_{t=1}^{+\infty} (1-p)^{t-1} = 1/p.$$

En effet, $\Pr(T \geq t) = (1-p)^{t-1}$ puisque $T \geq t$ si et seulement si les $t-1$ tirages donnent "face", et ces tirages sont indépendants. Soit d le degré de P. On supposera que $m \geq d+1$, sinon P pourrait être identiquement nul sur $[m]^s$ (voir question (B3)). On se trouve dans le jeu de pile ou face avec $p \geq 1 - d/m$, donc le nombre moyen de tirages est majoré par $m/(m-d)$.

(C2) Soit $G_{i,j}$ le graphe obtenu à partir de G en enlevant les sommets u_i et v_j, et toutes les arêtes incidentes. Soit existence(G, m) l'algorithme probabiliste qui teste l'existence d'un mariage parfait en tirant un point au hasard dans $[m]^{n^2}$. Voici l'algorithme cherché:

si $\neg$ existence(G, m) alors renvoyer "pas de mariage parfait", sinon:
  $M := \emptyset$
  pour i de 1 à n
    répéter
      choisir j tel que $(u_i, v_j) \in E$
      jusqu'à existence$(G_{i,j}, m')$
    $M := M \cup \{(u_i, v_j)\}$
    $G := G_{i,j}$
  finpour
  renvoyer M.

Commentaire: s'il n'y a pas de mariage parfait l'algorithme s'en aperçoit toujours. S'il y en a un, l'algorithme peut ne pas s'en apercevoir. Comme on l'a vu, la

probabilité d'erreur est majorée par n/m et peut être rendue aussi petite qu'on veut en prenant m suffisamment grand (ou en répétant le test plusieurs fois).
Si l'algorithme prend la branche "sinon", il existe forcément un mariage parfait, et l'algorithme finit par en renvoyer un avec probabilité 1 (mais dans le pire des cas, qui est de probabilité nulle, l'algorithme ne s'arrête pas; de même, dans une suite de tirages à pile ou face, le temps pour obtenir un tirage "pile" est infini dans le pire des cas). "choisir j" examine à tour de rôle tous les sommets de V reliés à i. Noter qu'on peut devoir examiner un sommet plusieurs fois avant de le retenir puisque existence($G_{i,j}, m'$) peut retourner un résultat erroné!
Analyse en moyenne: soit δ le degré de G. En prenant par exemple $m' = 2n$, le nombre moyen de tirages dans la boucle pour est en $O(n\delta)$: il faut construire les n arêtes de M, pour chacune de ces arêtes nous avons le choix entre $\leq \delta$ arêtes de G, et l'arête retenue est examinée en moyenne au plus $m'/(m'-n) = 2$ fois d'après la question précédente. On a donc bien un nombre moyen de tirages en $O(n\delta)$, puisqu'il y a δ tirages entre deux tests existence($G_{i,j}, m'$) sur la même arête j.

7 Processus séquentiels et bisimulation

On se donne un alphabet A de symboles, que l'on nomme souvent par des chaînes de caractères suggestives et que l'on considère comme des actions. On note $A^\star$ l'ensemble des mots écrits avec l'alphabet A. On appelle système de transition étiqueté ou LTS (pour "labelled transition system") un automate fini sans état initial ni ensemble d'états finaux. C'est donc la donnée d'un ensemble Q d'états, et d'un ensemble de triplets de $Q \times A \times Q$ notés $q_1 \xrightarrow{a} q_2$. Si $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \cdots q_n \xrightarrow{a_n} q_{n+1}$, et si u est le mot $a_1 \cdots a_n$ (ou la suite des actions $a_1, \ldots, a_n$), on note $q_1 \xrightarrow{u} q_{n+1}$.
Si on fixe un état q, on définit le langage $L(q)$ accepté par le LTS à partir de q par:

$$L(q) = \{u \in A^\star \mid \exists\, q' \in Q \;\; q \xrightarrow{u} q'\} \ .$$

(En termes d'automates, $L(q)$ est le langage accepté par l'automate obtenu en choisissant q comme état initial et en prenant l'ensemble de tous les états comme ensemble d'états finaux.)
On utilisera la notation suivante pour décrire un LTS sous forme d'un système d'équations: pour chaque $q \in Q$, on écrit $q = \sum_{\{(q,a,q') \mid q \xrightarrow{a} q'\}} a \cdot q'$. S'il n'y a pas de transition depuis q, on note $q = 0$ (somme nulle), ou plus simplement on omet de mettre une équation pour q. On dit que chaque état q est un *processus séquentiel*.

(A1) On considère l'alphabet $A = \{2F, \mathit{THE}, \mathit{CAFE}\}$. On considère le LTS suivant (juxtaposition de deux LTS, l'un disposant des états $p_1, p_2, \ldots$, l'autre des états $q_1, q_2, \ldots$):

$$\begin{array}{ll} p_1 = 2F \cdot p_2 & \quad q_1 = 2F \cdot q_2 + 2F \cdot q_3 \\ p_2 = \mathit{THE} \cdot p_1 + 2F \cdot p_3 & \quad q_2 = 2F \cdot q_4 \\ p_3 = \mathit{CAFE} \cdot p_1 & \quad q_3 = \mathit{THE} \cdot q_1 \\ & \quad q_4 = \mathit{CAFE} \cdot q_1 \end{array}$$

Comparer $L(p_1)$ et $L(q_1)$.

(A2) A-t-on envie de considérer p_1 et q_1 comme équivalents? Et que dire si l'on ajoute $q_5 = 0$, avec maintenant $q_2 = 2F \cdot q_4 + 2F \cdot q_5$?

Preuve. (A1) Si X et Y sont des ensembles de mots, on note $XY = \{uv \mid u \in X \text{ et } v \in Y\}$, $X^0 = \{\epsilon\}$, $X^{n+1} = X^n X$ et $X^\star = \bigcup_{n \geq 0} X^n$. On a:

$$L(p_1) = \{2F\,THE, 2F2F\,CAFE\}^\star\{\epsilon 2F, 2F\,THE, 2F2F, 2F2F\,CAFE\} = L(q_1)\ .$$

(A2) Les noms des actions sont suggestifs de distributeurs de boissons. On peut considérer que les processus p_1 et q_1 décrivent deux distributeurs. Dans le premier, on glisse 2F, après quoi l'utilisateur a la choix entre obtenir un thé ou insérer 2F supplémentaires, qui donnent droit à un café. Le second distributeur est moins interactif. Après que l'utilisateur a mis 2F, c'est lui qui décide s'il délivre un thé ou s'il demande 2F supplémentaires. S'il fait le second choix, il délivre le café après avoir reçu ces 2F. Les utilisateurs préfèreront généralement le premier distributeur. Le processus q_1 est enocre pire dans le LTS avec q_5 puisque le distributeur peut décider de ne rien délivrer après avoir reçu deux fois 2F. CQFD

La bisimulation est un outil qui permet de différencier des états tels que les états p_1 et q_1 de la question précédente. Une simulation est une relation $S \subseteq Q \times Q$ telle que

$$\forall\, p, p', q, a \ \ (p\,S\,q \text{ et } p \xrightarrow{a} p' \Rightarrow (\exists\, q' \ \ q \xrightarrow{a} q' \text{ et } p'\,S\,q'))\ .$$

On dit que q simule p s'il existe une simulation S telle que $p\,S\,q$.
Si S est une relation, on note $S^{-1} = \{(q,p) \mid p\;S\;q\}$. Une relation S telle que S et S^{-1} sont des simulations est appelée une bisimulation. On dit que p et q sont bisimilaires s'il existe une bisimulation S telle que $p\,S\,q$.

(B1) On considère une version simplifiée de l'exemple de la question (A1), avec $A = \{a, b, c\}$ et

$$\begin{array}{ll} p_1 = a \cdot p_2 & q_1 = a \cdot q_2 + a \cdot q_3 \\ p_2 = b \cdot p_3 + c \cdot p_4 & q_2 = b \cdot q_4 \\ & q_3 = c \cdot q_5 \end{array}$$

Montrer que p_1.simule q_1 et que q_1 ne simule pas p_1.

(B2) Est-ce équivalent de dire

- que p et q sont bisimilaires,
- que p simule q et q simule p?

PREUVE. (B1) Pour montrer que p_1 simule q_1, on construit S en commençant par y mettre le couple (q_1, p_1). Puisque l'action a est possible à partir de q_1, on est conduit à ajouter dans S les couples (q_2, p_2) et (q_3, p_2). Puis, en se plaçant en q_2 (q_3), on ajoute (q_4, p_3) $((q_5, p_4))$. Au total on a obtenu la simulation S suivante, qui prouve que p_1 simule q_1:

$$q_1 \, S \, p_1 \quad q_2 \, S \, p_2 \quad q_3 \, S \, p_2 \quad q_4 \, S \, p_3 \quad q_5 \, S \, p_4 \; .$$

Pour montrer que q_1 ne simule pas p_1, on raisonne par l'absurde. Supposons que T est une simulation telle que $p_1 \, T \, q_1$. Alors T doit contenir au moins l'un des deux couples (p_2, q_2) ou (p_2, q_3). Supposons que $p_2 \, T \, q_2$ $(p_2 \, T \, q_3)$. Les deux actions b et c sont possibles à partir de p_2, tandis que seule l'action b (c) est possible à partir de q_2 (q_3): contradiction.

(B2) Il est clair que si p et q sont bisimilaires, alors l'un simule l'autre et réciproquement. Le contraire n'est pas évident, puisqu'il faut utiliser une seule relation pour prouver la bisimulation. Et de fait le contre-exemple suivant (cf. la variante avec q_5 dans la question (A2)) montre que la bisimilarité est strictement plus forte que la simulation mutuelle:

$$\begin{array}{ll} p_1 = a \cdot p_2 + a \cdot p_4 & \quad q_1 = a \cdot q_2 \\ p_2 = b \cdot p_3 & \quad q_2 = b \cdot q_3 \end{array}$$

Supposons que T une simulation contenant (p_1, q_1). On a:

$$p_1 \, T \, q_1 \text{ et } p_1 \xrightarrow{a} p_4 \quad \Rightarrow \quad p_4 \, T \, q_2$$

Ceci implique que T^{-1} ne peut pas être une simulation, car l'action b ne peut pas être exécutée à partir de p_4. Donc p_1 et q_1 ne sont pas bisimilaires. Mais les simulations T et S suivantes prouvent que ces états se simulent réciproquement:

$$\begin{array}{llll} p_1 \, T \, q_1 & p_4 \, T \, q_2 & p_2 \, T \, q_2 & p_3 \, T \, q_3 \\ q_1 \, S \, p_1 & q_2 \, S \, p_2 & q_3 \, S \, p_3 \; . & \end{array}$$

Vu le commentaire fait dans la correction de la question (A2), la notion de bisimilarité est plus importante que la notion de simulation mutuelle. CQFD

On développe maintenant un exemple intéressant de processus séquentiel: un ordonnanceur. On suppose qu'il y a n tâches qui se répètent et qui n'interagissent avec le monde extérieur que pour recevoir leur top de départ a_i et émettre leur top de fin d'exécution b_i. Donc l'action b_i ne peut intervenir qu'après l'action a_i, et une nouvelle action a_i ne peut intervenir qu'après l'action b_i. Les tâches peuvent s'exécuter en parallèle, mais le top a_{i+1} ne peut être délivré par l'ordonnanceur qu'après qu'il ait donné le top a_i.

(C1) Ecrire un processus séquentiel qui met en oeuvre cette spécification informelle. Montrer que ce processus n'a jamais de deadlock, i.e., qu'il n'y a pas d'état à partir duquel aucune action n'est possible.

(C2) On passe maintenant à un cadre concurrent. On suppose que l'alphabet A est l'union disjointe d'un ensemble E d'actions externes et de deux ensembles I et $\overline{I}$ d'actions internes, qui sont en correspondance biunivoque par une bijection notée $^-$. On utilise $a, b, \alpha, \beta, \overline{\alpha}, \overline{\beta}, \ldots$ pour désigner les éléments de E, I et $\overline{I}$, respectivement. On appellera processus concurrent un ensemble (fini) de processus séquentiels. Ils ont le droit de réaliser indépendamment les uns des autres des transitions qui portent le nom d'une action externe. Les actions internes doivent être exécutées par paires: une action α d'un processus avec une action $\overline{\alpha}$ d'un autre processus, ce qui est formalisé par les règles suivantes (noter que la transition synchronisée ne porte pas de nom, l'idée étant qu'il s'agit d'une transition interne, non visible de l'extérieur):

$$\frac{p \xrightarrow{a} p'}{\{p, p_1, \ldots, p_n\} \xrightarrow{a} \{p', p_1, \ldots, p_n\}} \quad \frac{p \xrightarrow{\alpha} p' \qquad q \xrightarrow{\overline{\alpha}} q'}{\{p, q, p_1, \ldots, p_n\} \longrightarrow \{p', q', p_1, \ldots, p_n\}}$$

Ecrire un ordonnanceur répondant à la même spécification que ci-dessus, constitué d'un ensemble de n processus séquentiels, chacun ayant en charge de communiquer avec l'une des n tâches (ainsi qu'avec ses processus voisins).

(C3) Formuler une notion d'́equivalence entre le processus séquentiel de la question (C1) et le processus concurrent de la question (C2).

Preuve. (C1) Le choix de l'ensemble d'états est guidé par l'ensemble des actions possibles à un moment donné: on peut terminer des tâches, ce qui nécessite de tenir le compte de l'ensemble des tâches actives, et l'on peut activer la tâche suivante, ce qui nécessite de noter à qui est le prochain tour. De plus, cette tâche ne peut être activée que si elle a signalé qu'elle avait terminé son cycle précédent (d'où les deux cas dans les équations ci-dessous). D'où des états de la forme $q_{X,i}$, où $i \in \{1, \ldots, n\}$ et $X \subseteq \{1, \ldots, n\}$. Les transitions sont données par les équations suivantes:

$$q_{X,i} = \begin{cases} \sum_{j \in X} b_j \cdot q_{X \setminus \{j\}} + a_i \cdot q_{X \cup \{i\}, i+1} & (\text{si } i \notin X) \\ \sum_{j \in X} b_j \cdot q_{X \setminus \{j\}} & (\text{si } i \in X) \end{cases}$$

(C2) Si l'on localise le travail d'ordonnancment, chaque processus séquentiel en charge d'une tâche i doit avoir a_i et b_i comme actions, qui seront des actions externes du processus concurrent, et deux actions α_i et $\overline{\alpha_{i-1}}$ (modulo n) pour la synchronisation avec les voisins, qui seront des actions internes du processus concurrent: α_i se produit après a_i et en même temps que l'action $\overline{\alpha_i}$ du processus voisin, laquelle permet que a_{i+1} se produise. Donc, en ramenant au processus i, $\overline{\alpha_{i-1}}$ donne l'autorisation de déclencher a_i. Cette spécification est assez lâche, et laisse une certaine liberté quant à l'écriture des processus séquentiels locaux. On peut être tenté de travailler avec quatre états q_i^1, q_i^2, q_i^3 et q_i^4, et une seule transition d'un état à l'autre: $q_i^1 = a_i \cdot q_i^2$, $q_i^2 = \alpha_i \cdot q_i^3$, $q_i^3 = b_i \cdot q_i^4$ et $q_i^4 = \overline{\alpha_{i-1}} \cdot q_i^1$, et de définir l'ordonnanceur comme le processus concurrent $\{q_1^1, q_2^4, q_3^4, \ldots, q_n^4\}$. Examinons quelques transitions:

$$\begin{array}{rcl} \{q_1^1, q_2^4, q_3^4, \ldots, q_n^4\} & \xrightarrow{a_1} & \{q_1^2, q_2^4, q_3^4, \ldots, q_n^4\} \\ & \longrightarrow & \{q_1^3, q_2^1, q_3^4, \ldots, q_n^4\} \end{array}$$

d'où l'on peut exécuter b_1 ou a_2. Mais si l'on choisit d'exécuter en cascade toutes les synchronisations qui permettent d'activer $a_2, \ldots a_n$, on arrive au processus concurrent $\{q_1^3, \ldots, q_{n-1}^3, q_n^2\}$, à partir duquel il est impossible d'exécuter b_n sans avoir au préalable exécuté b_1: en effet b_1 est un verrou pour la synchronisation entre α_n et $\overline{\alpha_n} = \overline{\alpha_0}$, synchronisation qui est elle-même un verrou pour l'action b_n. Or l'ordonnanceur ne doit pas imposer d'ordre de terminaison entre tâches. Il faut donc introduire un peu de non-déterminisme dans les processus séquentiels locaux, pour éviter cette causalité entre b_1 et $\overline{\alpha_n}$, et, cycliquement, entre b_i et $\overline{\alpha_{i-1}}$. Il y a maintenant cinq états:

$$q_i^1 = a_i \cdot q_i^2 \quad q_i^2 = \alpha_i \cdot q_i^3 \quad q_i^3 = b_i \cdot q_i^4 + \overline{\alpha_{i-1}} \cdot q_i^5 \quad q_i^4 = \overline{\alpha_{i-1}} \cdot q_i^1 \quad q_i^5 = b_i \cdot q_i^1 \ .$$

Le processus concurrent qui code l'ordonnanceur est toujours $\{q_1^1, q_2^4, q_3^4, \ldots, q_n^4\}$, d'où l'on peut toujours atteindre $\{q_1^3, \ldots, q_{n-1}^3, q_n^2\}$, état du système à partir duquel on peut peut maintenant exécuter b_n indépendamment de b_1.

$$\begin{array}{rcl} \{q_1^3, \ldots, q_{n-1}^3, q_n^2\} & \longrightarrow & \{q_1^5, \ldots, q_{n-1}^3, q_n^3\} \\ & \xrightarrow{b_n} & \{q_1^5, \ldots, q_{n-1}^3, q_n^4\} \end{array}$$

(C3) Une manière d'être certain que le processus concurrent auquel on est arrivé dans la solution de la question (C2) est conforme à la spécification est de prouver qu'il est équivalent à $q_{\emptyset,1}$ tel que défini dans la solution de la question

(C1). La notion d'equivalence qui convient est une version affaiblie de la notion de bisimilarité, appelée bisimilarité faible. L'important est que ce soient les mêmes actions externes qui soient possibles (et ce héréditairement) à partir d'états faiblement bisimilaires, quitte à devoir attendre quelques transitions internes. Une simulation contient une paire (q séquentiel, $\{q_1, \ldots, q_n\}$ concurrent) si pour tous q', a tels que $q \xrightarrow{a} q'$, il existe une suite de transitions depuis $\{q_1, \ldots, q_n\}$ dont une seule est externe (et porte le nom a) aboutissant à un état $\{q'_1, \ldots, q'_n\}$ tel que $q' S \{q'_1, \ldots, q'_n\}$. Dans l'autre sens, une simulation contient une paire ($\{q_1, \ldots, q_n\}$,q) si à toute transition externe $\{q_1, \ldots, q_n\} \xrightarrow{a} \{q'_1, \ldots, q'_n\}$ (un seul des q'_i différent de q_i) correspond une transition $q \xrightarrow{a} q'$ telle que $\{q'_1, \ldots, q'_n\}\, S\, q'$, et si pour toute transition interne $\{q_1, \ldots, q_n\} \longrightarrow \{q'_1, \ldots, q'_n\}$, on a $\{q'_1, \ldots, q'_n\}\, S\, q$. (En réalité, les processus séquentiels sont un cas particulier de processus concurrents, sans actions internes, et la définition générale de bisimilarité faible est entre processus. Les processus sont eux-mêmes plus généraux que ce que nous avons appelé ici processus concurrents.)
Avec cette définition, on peut vérifier formellement (long mais sans difficulté) que $\{q_1^1, q_2^4, q_3^4, \ldots, q_n^4\}$ et $q_{\emptyset,1}$ sont faiblement bisimilaires. CQFD

8 Graphes série-parallèles

On note $\#X$ le cardinal d'un ensemble X. Un hypergraphe est un couple (G, Γ), où G est un ensemble de sommets et $\Gamma \subseteq \{X \subseteq G \mid \#X \geq 2\}$. Chaque élément de Γ est appelé une hyperarête. On note $\underline{\Gamma} = \Gamma \cup \{\{x\} \mid x \in G\}$. Un graphe (non orienté) est donc le cas particulier d'hypergraphe dans lequel les hyperarêtes ont toutes exactement deux éléments. Si (G, Γ) est un hypergraphe et $G' \subseteq G$, l'hypergraphe $(G', \{X \in E \mid X \subseteq G'\})$ est appelé l'hypergraphe induit sur G'. (Noter qu'un hypergraphe induit par un graphe est un graphe, on parle alors de graphe induit.) Si (G, E) est un graphe, on note, pour $a, b \in G$:

$$a \mathrel{\substack{\frown \\ \smile}} b \Leftrightarrow (a = b \text{ ou } \{a, b\} \in E) .$$

A tout graphe (G, E), on associe l'hypergraphe $(G, \hat{E})$, où $\hat{E}$ est l'ensemble des parties connexes de G de cardinal ≥ 2. A tout hypergraphe (G, Γ), on associe un graphe $(G, \check{\Gamma})$, où $\check{\Gamma} = \{X \in \Gamma \mid \#X = 2\}$. Si (G, Γ) est un hypergraphe, on pose $\Gamma^{\perp} = \{X \subseteq G \mid \#X \geq 2 \text{ et } X \notin \Gamma\}$. Un hypergraphe parallèle est un hypergraphe tel que

$$(X_1 \in \Gamma, X_2 \in \Gamma \text{ et } X_1 \cap X_2 \neq \emptyset) \Rightarrow X_1 \cup X_2 \in \Gamma .$$

On considère le graphe suivant:

$$G(a, b, c, d) = (\{a, b, c, d\}, \{\{a, b\}, \{b, c\}, \{c, d\}\}) .$$

Un graphe série-parallèle est un graphe G tel qu'il n'existe aucun sous-ensemble $G' = \{a, b, c, d\}$ de G tel que le graphe induit sur G' soit $G(a, b, c, d)$. Le but de ce sujet est d'obtenir différentes caractérisations des graphes série-parallèles.

(A1) Montrer que dans un graphe série-parallèle (G, E), un ensemble X d'arêtes est connexe si et seulement si

$$\forall\, a, b \in X \;\; \exists\, c \in X \;\; a \mathrel{\substack{\frown \\ \smile}} c \text{ et } c \mathrel{\substack{\frown \\ \smile}} b .$$

(A2) Soit (G, E) un graphe série-parallèle. Montrer que $(G, \hat{E})$ et $(G, \hat{E}^{\perp})$ sont des hypergraphes parallèles.

Preuve. (A1) Un X satisfaisant la condition est a fortiori connexe; Réciproquement, on procède par récurrence sur la longueur d'un chemin de $a = c_0, c_1, \ldots,$

$c_n = b$ (avec $\{c_i, c_{i+1}\} \in E$). On peut donc supposer qu'il existe c tel que $a \overset{\frown}{\smile} c$ et $c \overset{\frown}{\smile} c_{n-1}$. Si $a = c$ ou $c = c_{n-1}$, alors c_{n-1} convient. Si $\{a, c\} \in E$ et $\{c, c_{n-1}\} \in E$, on considère le graphe induit sur $\{a, c, c_{n-1}, b\}$. Ce graphe ne peut pas être $G(a, c, c_{n-1}, b)$. Donc au moins l'une des arêtes $\{a, b\}$, $\{a, c_{n-1}\}$ ou $\{c, b\}$ est dans E, ce qui dans chaque cas donne un chemin d'au plus deux arêtes entre a et b.

(A2) Pour $\hat{E}$, cela résulte immédiatement de la définition de connexité: si X_1 et X_2 sont connexes, si $a \in X_1 \cap X_2$, $a_1 \in X_1$ et $a_2 \in X_2$, on obtient un chemin de a_1 à a_2 dans $X_1 \cup X_2$ en mettant bout à bout un chemin de a_1 à a dans X_1 et un chemin de a à a_2 dans X_2. Supposons maintenant que X_1 et X_2 ne sont pas connexes (ce qui implique que chacun a au moins deux éléments) et ont une intersection non vide. Supposons par l'absurde que $X_1 \cup X_2$ est connexe. Soit $a \in X_1 \cap X_2$. Soit $a_1 \in X_1$ différent de a et non relié à a par un chemin dans X_1 (il existe un tel a car sinon X_1 serait connexe). Par (A1), il existe $c_1 \in X_1 \cup X_2$ tel que $a \overset{\frown}{\smile} c_1$ et $c_1 \overset{\frown}{\smile} a_1$. Ce c_1 est forcément dans $X_2 \setminus X_1$, car sinon a, c_1, a_1 serait un chemin da a à a_1 dans X_1. De même on peut trouver $a_2 \in X_2$ différent de a et $c_2 \in X_1 \setminus X_2$ tels que $a \overset{\frown}{\smile} c_2$, $c_2 \overset{\frown}{\smile} a_2$, et a_2 n'est pas relié à a par un chemin dans X_2. Le graphe induit sur $\{c_2, a, c_1, a_1\}$ est différent de $G(c_2, a, c_1, a_1)$. Donc l'une des arêtes $\{c_2, a_1\}$, $\{a, a_1\}$ ou $\{c_2, c_1\}$ est dans E. Les deux premiers cas sont exclus car ils contredisent la non-existence d'un chemin de a à a_1 dans X_1. Donc $\{c_2, c_1\} \in E$. On considère alors le graphe induit sur $\{a_2, c_2, c_1, a_1\}$, qui doit être différent de $G(a_2, c_2, c_1, a_1)$, d'où l'on déduit similairement que $\{a_1, a_2\}$ est dans E. Mais alors le graphe induit sur $\{a, c_1, a_1, a_2\}$ est $G(a, c_1, a_1, a_2)$, car aucune des arêtes $\{a_2, c_1\}$, $\{a, a_1\}$, $\{a, a_2\}$ n'est dans E: contradiction. CQFD

La question précédente suggère la définition suivante: un hypergraphe série est un hypergraphe (G, Γ) tel que $(G, \Gamma^\perp)$ est parallèle. Donc, on a montré que si (G, E) est un graphe série-parallèle, alors $(G, \hat{E})$ est un hypergraphe série et parallèle. On montre maintenant que la structure de tout hypergraphe série et parallèle peut être décrite à partir d'un arbre dont les feuilles sont les sommets de l'hypergraphe.

(B1) Soit (G, Γ) un hypergraphe série, et soit $X \in \Gamma$. Montrer qu'il existe un ensemble unique $\{X_1, \ldots, X_n\} \subseteq \underline{\Gamma^\perp}$ ($n \geq 2$) tel que les X_i sont disjoints deux à deux, $X = \bigcup_{i=1}^{i=n} X_i$, et pour tout $Y \subseteq X$, si Y rencontre au moins deux X_i, alors $Y \in \Gamma$. On note $X = \mathbf{P}(X_1, \ldots, X_n)$. Montrer une décomposition similaire $X = \mathbf{O}(X_1, \ldots, X_n)$ pour $X \in \Gamma^\perp$ pour un hypergraphe parallèle.

(B2) Utiliser (B1) pour associer à tout hypergraphe série et parallèle un arbre construit avec des sommets intérieurs étiquetés par **O** et **P** et des feuilles étiquetées par les éléments de G (sans répétition). Que peut-on dire des positions occupées par des sommets **P** et **O**?

(B3) On considère l'arbre obtenu en (B2) comme un ordre partiel (par la relation $n_1 \leq n_2$ si n_1 est sur le chemin de la racine à n_2. Deux sommets quelconques n_1 et n_2 ont un plus grand minorant commun $n_1 \wedge n_2$ (qui est le sommet où les chemins de la racine à n_1 et n_2 se séparent). On note $\bigwedge X$ le plus grand commun minorant d'un ensemble (fini) X de sommets. Montrer l'équivalence suivante, pour tout $X \subseteq G$ (identifié avec l'ensemble des feuilles étiquetées par les éléments de X):

$$X \in \Gamma \Leftrightarrow \bigwedge X \text{ a l'étiquette } \mathbf{P} \, .$$

Preuve. (B1) On définit les X_i comme les sous-ensembles maximaux de X qui sont dans $\underline{\Gamma^\perp}$. Il sont disjoints par maximalité et parce que par hypothèse $(G, \Gamma^\perp)$ est parallèle. On déduit facilement que l'union des X_i couvre X du fait que $\underline{\Gamma^\perp}$ contient tous les singletons. Il y a au moins deux X_i, car chaque X_i est différent de X: en effet, soit $\#X_i = 1$, alors que $\#X \geq 2$, soit $X_i \in \Gamma^\perp$, alors que $X \in \Gamma$. Supposons $i \neq j$, $Y \cap X_i \neq \emptyset$, $Y \cap X_j \neq \emptyset$, et $Y \notin \Gamma$. On a $Y \in \Gamma^\perp$ (car Y a au moins deux éléments), donc $Y \cup X_i \in \Gamma^\perp$, ce qui par maximalité implique $Y \subseteq X_i$, et contredit $Y \cap X_j \neq \emptyset$.

(B2) On procède par récurrence sur $\#G$. Si $\#G = 1$, l'arbre associé est réduit à une feuille. Si $\#G \geq 2$, alors $G \in \Gamma$ ou $G \in \Gamma^\perp$. Dans le premier cas, par (B1), on a $G = \mathbf{P}(X_1, \ldots, X_n))$, ce qui fournit la racine de l'arbre. Comme $\#X_i < \#X$, on peut appliquer la récurrence à chaque X_i (on montre facilement que les hypergraphes induits sur les sous-ensembles héritent des qualités série et parallèle). Si $G \in \Gamma^\perp$, alors le raisonnement est identique, la racine de l'arbre étant maintenant étiquetée **O**. On note que lorsqu'on applique la récurrence à X_i, il n'y a plus de choix: c'est un sommet **O** (**P**) si la racine est **P** (**O**). L'arbre présente donc une alternance de sommets **P** et **O**. Une branche de l'arbre s'arrête lorsqu'elle a atteint un singleton.

(B3) On va prouver les deux implications suivantes (pour X de cardinal ≥ 2), qui ensemble montrent immédiatement l'équivalence de l'énoncé:

$$\begin{array}{ll} \bigwedge X \text{ a l'étiquette } \mathbf{P} & \Rightarrow X \in \Gamma \\ \bigwedge X \text{ a l'étiquette } \mathbf{O} & \Rightarrow X \in \Gamma^\perp \, . \end{array}$$

On montre la première implication (la preuve de la seconde étant symétrique). En plus des étiquettes **P** ou **O**, on peut garder sur chaque sommet intérieur l'information du sous-ensemble de Y qui lui correspond (G correspond à la racine, puis viennent au niveau 1 les X_i, etc...). Il est facile de voir (en redescendant des feuilles vers la racine) que l'ensemble Y ainsi associé à un sommet intérieur n est aussi l'ensemble des feuilles au-dessus de ce sommet. Donc, si Y est l'ensemble correspondant au sommet $\bigwedge X$, on a $X \subseteq Y$. Soient $Y_1, \ldots, Y_n$ les ensembles associés aux fils de $\bigwedge X$. Par définition de $\bigwedge$, il doit y avoir un embranchement à ce sommet, i.e., il doit y avoir deux éléments distincts de X dont les chemins vers la racine passent par deux Y_i différents, i.e., $(\exists\, i_1, i_2 \;\; i_1 \neq i_2, X \cap Y_{i_1} \neq \emptyset$ et $X \cap Y_{i_2} \neq \emptyset)$. On conclut alors par (B1).

CQFD

La question précédente suggère de définir des arbres série-parallèle (ASP) comme suit. On fixe au départ un ensemble G.

$$\frac{a \in G}{a \ \mathrm{ASP}(\{a\})}$$

$$\frac{s_1 \ \mathrm{ASP}(X_1) \quad \cdots \quad s_n \ \mathrm{ASP}(X_n)}{\mathbf{O}(s_1, \ldots, s_n) \ \mathrm{ASP}(X_1 \cup \cdots \cup X_n)} \qquad \frac{s_1 \ \mathrm{ASP}(X_1) \quad \cdots \quad s_n \ \mathrm{ASP}(X_n)}{\mathbf{P}(s_1, \ldots, s_n) \ \mathrm{ASP}(X_1 \cup \cdots \cup X_n)}$$

Dans les deux dernières règles, on suppose $n \geq 2$ et $\forall\, i, j \;\; (i \neq j \Rightarrow X_i \cap X_j = \emptyset)$. La notation $s \ \mathrm{ASP}(X)$ veut dire que s est un ASP dont les feuilles sont en bijection avec X (cf. preuve de (B3)); l'hypothèse $X_i \cap X_j$ garantit la non-répétition des étiquettes des feuilles. On dit que s est un ASP pour G, et on lui associe le graphe $(G, \overline{s})$, où $\{a_1, a_2\} \in \overline{s}$ si et seulement si $a_1 \wedge a_2$ a l'étiquette **P**. On note $(G, \overline{s^\perp})$ le graphe dont les arêtes sont exactement les ensembles $\{a, b\}$ qui ne sont pas dans $\overline{s}$. (La notation reflète le fait que ce graphe est bien le graphe associé à un ASP, celui obtenu en intervertissant les **P** et les **O** dans s.)

(C1) Montrer que si s est un ASP pour un ensemble G, alors les sous-ensembles connexes du graphe $(G, \overline{s})$ de cardinal ≥ 2 sont les sous-ensembles X (vus comme ensembles de feuilles de l'arbre) tels que $\bigwedge X$ a l'étiquette **P**.

(C2) Montrer que les seuls ensembles X à la fois connexes dans $(G, \overline{s})$ et $(G, \overline{s^\perp})$ sont les singletons.

(C3) Montrer que $(G, \overline{s})$ est série-parallèle.

(C4) Donner un exemple de graphe série-parallèle pour lequel la relation $\frown\!\!\!\!\!\!\smile$ n'est pas transitive (i.e., dans (A1), les chemins de longueur 2 sont bien nécessaires en général).

Preuve. (C1)($\Rightarrow$) On montre d'abord une propriété générale d'un graphe quelconque (G, E).

(P) On peut énumérer toute partie connexe X (avec des répétitions éventuelles) par une suite $a_0, \ldots, a_m$ telle que $\{a_i, a_{i+1}\} \in E$ pour tout i.

Preuve de (P). On montre par récurrence sur la taille de $Y \subseteq X$ que l'on peut trouver une suite $a_0, \ldots, a_p$ telle que $Y \subseteq \{a_0, \ldots, a_p\} \subseteq X$ et $\{a_i, a_{i+1}\} \in E$ pour tout i. Si $\#Y = 2$, l'existence de cette suite résulte de ce que X est supposé connexe. Si $Y = Z \cup \{a\}$ ($a \notin Z$), on a par récurrence une suite $a_0, \ldots, a_p$ pour Z. Il existe par ailleurs un chemin de a_p à a dans X, par connexité de X. On conclut en concaténant.

Il suffit, par la propriété (P), de montrer que $a_0 \wedge \cdots \wedge a_m$ a l'étiquette **P**, pour toute suite $a_0, \ldots, a_m$ telle que $\{a_i, a_{i+1}\} \in E$ pour tout i, par récurrence sur la longueur de cette suite. Le cas $m = 2$ est la définition même de $\overline{s}$. Si $m > 2$, on pose $a = a_0 \wedge \cdots \wedge a_{m-1}$ et $a' = a_0 \wedge \cdots \wedge a_{m-1} \wedge a_m$. Par récurrence, on sait que a a l'étiquette **P**. Si $a' = a$, alors on a fini. Si $a' < a$, on montre que $a' = a_m \wedge a_i$ pour tout $i < m$, ce qui permet de conclure, en prenant en particulier $i = m - 1$. On a évidemment $a' \leq a_m \wedge a_i$. Prouvons que $a_m \wedge a_i < a$. Sinon, on aurait $a_m \wedge a_i \geq a$ (dans un arbre deux points majorés – ici par a_i – sont comparables), qui implique $a \leq a_m \wedge a_i \wedge a = a'$: contradiction. Donc $a_m \wedge a_i \leq a \wedge a_m = a'$.

(C1)($\Leftarrow$) Supposons que $\bigwedge X$ a l'étiquette **P**. Plus précisément, soit $\bigwedge X = \mathbf{P}(s_1, \ldots, s_n)$ ($n \geq 2$), où chaque s_i est un ASP pour Y_i. On peut alors partitionner X en $X_1, \ldots, X_n$ (disjoints) tels que $X_1 \subseteq Y_1, \ldots, X_n \subseteq Y_n$. Et l'on a $a \wedge b = \mathbf{P}(s_1, \ldots, s_n)$ pour tous $a \in X_i, b \in X_j$ ($i \neq j$). Ceci montre la connexité de X: il existe une arête entre deux points dès qu'ils appartiennent à deux classes différentes de la partition, et des chemins de longueur 2 entre deux points quelconques a_1, a_2 d'une même classe: en fixant b dans une autre classe, on a en effet $\{a_1, b\} \in E$ et $\{a_2, b\} \in E$.

(C2) Par la caractérisation (C1), une partie X non singleton telle que dans l'énoncé vérifie à la fois que $\bigwedge X$ a l'étiquette **P** et l'étiquette **O**: contradiction.

(C3) Supposons que G contient un graphe induit $G(a, b, c, d)$. Ce graphe a la propriété que l'ensemble de ses quatre sommets est connexe dans $G(a, b, c, d)$ et dans le graphe dont les arêtes sont celles qui ne figurent pas dans $G(a,b,c,d)$. Une telle situation a été exclue en (C2).

(C4) Soit le graphe $(\{a,c,b\}, \overline{\mathbf{P}(\mathbf{0}(a,b),c)})$, On a $a \smile c$ et $c \smile b$, mais $\{a,b\}$ n'est pas une arête du graphe. CQFD

On résume les résultats obtenus sous la forme d'une dernière question.

Montrer que les propriétés suivantes sont équivalentes pour un graphe (G,E):

(D1) (G,E) est série-parallèle.

(D2) Les seuls ensembles à la fois connexes dans (G,E) et dans $(G, \{\{a,b\} \mid \{a,b\} \notin E\}$ sont les singletons.

(D3) $(G,\hat{E})$ est série et parallèle; de plus, cette correspondance a pour inverse la correspondance $(G,\Gamma) \mapsto (G,\check{\Gamma})$.

Preuve. On a (D1)⇒(D3)⇒(D2)⇒(D1). On a prouvé en (A) que $(G,\hat{E})$ est série et parallèle. Inversement, si (G,Γ) est série et parallèle, on a construit en (B) un arbre associé $s(\Gamma)$ offrant une caractérisation des hyperarêtes de G, puis on a défini un graphe associé $(G,\overline{s(\Gamma)})$. La lecture de ces caractérisation et définition donne $\overline{s(\Gamma)} = \check{\Gamma}$, et l'on a donc (D2), qui n'est autre que (C2) pour $s(\Gamma)$. $(G,\check{\Gamma})$ est donc série-parallèle par (C3). L'égalité $\check{\hat{E}} = E$ est évidente. L'égalité $\hat{\check{\Gamma}} = \Gamma$ résulte de (B3) et (C1). CQFD

9 Unification

Le problème de l'unification est celui de trouver, étant donnés deux termes t_1 et t_2, une substitution σ telle que $\sigma(t_1) = \sigma(t_2)$. On note le problème $(t_1 \stackrel{?}{=} t_2)$, et on dit que σ est une solution ou un unificateur de l'équation. On suppose donnés un ensemble V de variables (infini), et un ensemble (fini) F (disjoint de V) de symboles de fonctions (ou constructeurs) f, chacun muni d'une arité $a(f) \in \omega$ (ω désigne l'ensemble des entiers naturels). Les termes sont construits à partir des variables et des symboles de fonctions, en respectant les arités:

$$t ::= x | f(t_1, \ldots, t_{a(f)}) \; .$$

(En particulier, si a est une constante, i.e., un symbole d'arité 0, alors a est un terme.) Une substitution est une fonction σ de l'ensemble des variables dans l'ensemble des termes telle que $\{x \mid \sigma(x) \neq x\}$ est fini. Elle s'étend en une fonction de l'ensemble des termes dans l'ensemble des termes: $\sigma(f(t_1, \ldots, t_n)) = f(\sigma(t_1), \ldots \sigma(t_n))$.
Une autre présentation des termes est fournie via la notion d'occurrence. Une occurrence est un mot $u \in \omega^\star$, i.e., une suite d'entiers naturels. La suite vide est notée ϵ. Un terme est une fonction partielle de $\omega^\star$ dans $V \cup F$ satisfaisant certaines conditions.

(A1) Expliciter ces conditions, et les transformations inverses entre les deux définitions. Par exemple, écrire $s = f(g(a, h(x)), b)$ sous forme de fonction partielle.

(A2) Définir la notion de sous-terme t/u d'un terme t à l'occurrence u.

Preuve. (A1) Le meilleur guide à l'intuition est de définir d'abord la fonction associée à un terme t, fonction que l'on appelle également t. Si $t = x$ alors $t(\epsilon) = x$ et $t(u)$ est indéfini si $u \neq \epsilon$. Si $t = f(t_0, \ldots, t_{n-1})$, alors $t(\epsilon) = f$, $t(iu)$ est indéfini si $i \geq n$, et si $i < n$, alors $t(iu)$ est défini si et seulement si $t_i(u)$ est défini, et dans ce cas $t(iu) = t_i(u)$. Sur l'exemple s, on a:

$$s(\epsilon) = f \quad s(0) = g \quad s(00) = a \quad s(01) = h \quad s(010) = x \quad s(1) = b \; .$$

En examinant cette construction, les conditions suivantes apparaissent naturellement:

- $t(\epsilon)$ est défini.
- Si $t(u)$ est défini et v est un préfixe de u, alors $t(v)$ est défini.
- Si $t(u)$ est défini et si $t(u)$ est un symbole d'arité n, alors $t(ui)$ est défini si et seulement si $i \leq n-1$.

Pour se convaincre qu'elles sont suffisantes, on procède à la transformation inverse, qui associe un terme $\underline{t}$ à une fonction t. Si $t(\epsilon)$ est un symbole f d'arité n, on définit les fonctions $t_1, \ldots, t_n$ comme suit: $t_i(u)$ est défini si et seulement si $t(iu)$ est défini, et dans ce cas $t_i(u) = t(iu)$. On pose $\underline{t} = f(\underline{t_1}, \ldots, \underline{t_n})$. (La récurrence est sur le cardinal du domaine de définition de la fonction.)
Le détail de ce que les transformations sont inverses est laissé au lecteur. On peut noter que la deuxième condition sert à garantir que la transformation est bien surjective (et que partant d'une fonction, construisant le terme associé puis la fonction associée, on retrouve bien la même fonction).

(A2) On pose $t/\epsilon = t$, x/u est indéfini si $u \neq \epsilon$, et

$$f(t_0, \ldots, t_{n-1})/iu \text{ est défini si et seulement si } i \leq n-1 \text{ et } t_i/u \text{ est défini },$$

et dans ce cas $f(t_1, \ldots, t_n)/iu = t_i/u$. CQFD

(B1) Résoudre (positivement en fournissant une substitution) ou négativement (en montrant qu'ii n'y en a pas) les problèmes d'unification qui suivent.

$$\begin{array}{ll}
(1) & (f(g(a,x),h(y)) \stackrel{?}{=} f(z,g(x',a))) \\
(2) & (f(g(a,x),h(y)) \stackrel{?}{=} f(z,h(f(a,z)))) \\
(3) & (f(g(a,x),h(y)) \stackrel{?}{=} f(z,h(f(a,y)))
\end{array}$$

(B2) Formuler la notion d'unification d'un ensemble d'équations, et montrer que ce problème n'est pas plus général que celui avec une seule équation. Montrer que si un problème d'unification a une solution, alors il en a une infinité.

(B3) Formuler une procédure (non-déterministe, i.e., offrant plusieurs choix d'exécution) d'unification fondée sur la transformation progressive de l'équation de départ en ensembles d'équations "plus simples". Caractériser les états finaux de l'algorithme.

(B4) Discuter la complexité du problème d"unification suivant:

$$(x_1 \stackrel{?}{=} f(x_0, x_0)), (x_2 \stackrel{?}{=} f(x_1, x_1)), \ldots, (x_n \stackrel{?}{=} f(x_{n-1}, x_{n-1})) .$$

Donner une description compacte de la solution.

(B5) Adapter la procédure d'unification de telle sorte que l'exemple de la question précédente puisse être traité en temps linéaire, ou rester en temps exponentiel, selon l'ordre d'exécution choisi.

(B6) Montrer la correction de l'algorithme de la question précédente, i.e. qu'il termine toujours si le problème initial a une solution, et que la substitution qu'il spécifie est bien une solution. Que peut-on dire de plus de cette solution parmi toutes les autres solutions (cf. question (B2))?

Preuve. (B1) Le premier problème n'a pas de solution. En effet, une propriété importante des substitutions est la suivante:

$$\forall \sigma, t, u \ \ (t(u) \text{ défini et } t(u) \in F) \Rightarrow \sigma(t)(u) = t(u) .$$

Soient $t_1 = f(g(a, x), h(y))$ et $t_2 = f(z, g(x', a))$. On a $t_1(1) = h$ et $t_2(1) = g$. S'il y avait une sustitution σ telle que $\sigma(t_1) = \sigma(t_2)$, on aurait en particulier $\sigma(t_1)(1) = \sigma(t_2)(1)$. Mais $\sigma(t_1)(1) = h$ et $\sigma(t_2)(1) = g$.
Pour résoudre le second problème, on procède par réécriture. Le problème est équivalent au problème de trouver une substitution qui unifie à la fois $g(a, x)$ et z d'une part, et $h(y)$ et $h(f(a, z))$ d'autre part. Ce qui conduit à écrire:

$$\{(f(g(a, x), h(y)) \stackrel{?}{=} f(z, h(f(a, z))))\} \ \rightarrow \ \{(g(a, x) \stackrel{?}{=} z), (h(y) \stackrel{?}{=} h(f(a, z)))\} .$$

De là, on peut soit continuer à décomposer, et obtenir ainsi l'équation $(y \stackrel{?}{=} f(a, z))$, soit propager la valeur de z donnée par la première équation dans la seconde. En appliquant ces deux transformations, par exemple la seconde d'abord, on obtient:

$$\begin{aligned} &\{(g(a, x) \stackrel{?}{=} z), (h(y) \stackrel{?}{=} h(f(a, z)))\} \\ &\quad \rightarrow \ \{(g(a, x) \stackrel{?}{=} z), (h(y) \stackrel{?}{=} h(f(a, g(a, x))))\} \\ &\quad \rightarrow \ \{(g(a, x) \stackrel{?}{=} z), (y \stackrel{?}{=} f(a, g(a, x)))\} . \end{aligned}$$

On peut vérifier qu'il s'agit bien d'une substitution qui unifie les deux termes du deuxième exemple.

Pour le troisième exemple, on commence par:

$$\begin{array}{rcl}\{(f(g(a,x),h(y)) \stackrel{?}{=} f(z,h(f(a,y)))\} & \to & \{(g(a,x) \stackrel{?}{=} z),(h(y) \stackrel{?}{=} h(f(a,y)))\} \\ & \to & \{(g(a,x) \stackrel{?}{=} z),(y \stackrel{?}{=} f(a,y))\} .\end{array}$$

Ici, on utilise une autre propriété des substitutions. Un sous-terme strict d'un terme t est un terme $s = t/u$, avec $u \neq \epsilon$ (et t/u défini):

$$\forall \sigma, s, t \;\; (s \text{ sous-terme strict de } t \Rightarrow \sigma(s) \text{ sous-terme strict de } \sigma(t)) .$$

On conclut que le problème n'a pas de solution, puisque, dans l'équation $(y \stackrel{?}{=} f(a,y))$, y apparaît comme sous-terme strict de $f(a,y)$.

(B2) Soit $S = \{(s_1 \stackrel{?}{=} t_1),\ldots,(s_n \stackrel{?}{=} t_n)\}$. Le problème correspondant d'unification est de trouver une même substitution σ telle que $\sigma(s_1) = \sigma(t_1),\ldots,\sigma(s_n) = \sigma(t_n)$. On peut exprimer ce problème avec une seule équation, quitte à introduire un nouveau symbole s'il n'y a pas de symbole d'arité n: $(f(s_1,\ldots,s_n) \stackrel{?}{=} f(t_1,\ldots,t_n))$. Si σ est solution d'un problème d'unification, alors, pour toute substitution τ, $\tau \circ \sigma$ (où τ est l'extension de τ aux termes) est une solution. Si le problème a une solution, il en a donc une infinité.

(B3) Le système de transformation dont on a donné quelques exemples se formalise comme suit en toute généralité. On note $+$ l'union disjointe de deux ensembles, et on note $X + x$ au lieu de $X + \{x\}$. On note $[t/x]$ la substitution σ définie par $\sigma(x) = t$ et $\sigma(y) = y$ si $y \neq x$. On note $\sigma(S) = \{(\sigma(t_1) \stackrel{?}{=} \sigma(t_2)) \mid (t_1 \stackrel{?}{=} t_2) \in S\}$. On note $V(t)$ l'ensemble des variables apparaissant dans t, et $V(S) = \bigcup_{(t_1 \stackrel{?}{=} t_2) \in S}(V(t_1) \cup V(t_2))$.

$$Dec \quad \frac{S = S_1 + (f(s_1,\ldots,s_n) \stackrel{?}{=} f(t_1,\ldots,t_n)))}{S \;\to\; S_1 + \{(s_1 \stackrel{?}{=} t_1),\ldots,(s_n \stackrel{?}{=} t_n)\}}$$

$$Prop \quad \frac{S = S_1 + (x \stackrel{?}{=} t) \quad (x \in V(S_1) \setminus V(t))}{S \;\to\; [t/x](S_1) + (x \stackrel{?}{=} t)}$$

$$(\mathrm{A}) \quad \frac{S = S_1 + (t \stackrel{?}{=} x)}{S \;\to\; S_1 + (x \stackrel{?}{=} t)} \qquad\qquad (\mathrm{B}) \quad \frac{S = S_1 + (x \stackrel{?}{=} x)}{S \;\to\; S_1}$$

$$\frac{S = S_1 + (f(s_1,\ldots,s_n) \stackrel{?}{=} g(t_1,\ldots,t_m)) \quad (f \neq g)}{S \ \rightarrow \ \mathit{FAIL}}$$

$$\frac{S = S_1 + (x \stackrel{?}{=} t) \quad (x \in V(t))}{S \ \rightarrow \ \mathit{FAIL}}$$

S'il s'agit d'unifier t_1 et t_2, on commence l'exécution avec $\{(t_1 \stackrel{?}{=} t_2)\}$. Un état final du système est un état à partir duquel il n'y a plus de réécriture possible. Dans un état final $S \neq \mathit{FAIL}$, toutes les équations sont de la forme $x = t$, où $t \neq x$ (par les règles *Dec*, (A) et (B)), et de plus x n'apparaît en tout qu'une seule fois dans S (par la règle *Prop* et par la dernière règle). Ces équations définissent une substitution σ: $\sigma(x) = t$ si $(x \stackrel{?}{=} t) \in S$ et $\sigma(x) = x$ sinon. Et on a, pour tout $(x \stackrel{?}{=} t) \in S$, $\sigma(x) = t = \sigma(t)$ (puisque les seules variables que σ transforme sont les membres gauches de S, et n'apparaissent nulle part ailleurs, donc en particulier pas dans t). Donc σ est unificateur de S.

(B4) En appliquant la procédure d'unification dans un ordre ou dans un autre, on arrive à la substitution σ suivante:

$$\begin{array}{l}
\sigma(x_1) = f(x_0,x_0)\\
\sigma(x_2) = f(f(x_0,x_0),f(x_0,x_0))\\
\sigma(x_3) = f(f(f(x_0,x_0),f(x_0,x_0)),f(f(x_0,x_0),f(x_0,x_0)))\\
\vdots
\end{array}$$

La taille de la substitution est exponentielle, et donc a fortiori la durée d'exécution qui entre autres choses construit cette substitution. Cependant, le problème de départ peut être considéré comme une représentation de la solution, pour peu qu'on en énumère les équations dans un ordre adéquat! On a en effet:

$$\sigma = [f(x_0,x_0)/x_1] \circ [f(x_1,x_1)/x_2] \circ \cdots \circ [f(x_{n-1},x_{n-1})/x_n] \ .$$

(B5) L'analyse faite dans la question précédente suggère de construire l'unificateur comme une composition (non calculée) de substitutions. Ceci conduit à maintenir une liste d'équations tout au long de l'algorithme. On réécrit désormais des paires $T;S$, où S est un ensemble d'équations et où T est une liste d'équations de la forme $(x \stackrel{?}{=} t)$ (avec $x \notin V(t)$). On note $(x \stackrel{?}{=} t) :: T$ la liste obtenue en ajoutant l'élément $(x \stackrel{?}{=} t)$ en tête de la liste T. La seule modification réelle par rapport au système de la question (B3) est que dans la règle *Prop*, on range désormais l'équation $(x \stackrel{?}{=} t)$, ce qui la met à l'abri de futures substitutions par des règles *Prop* ultérieures (voir l'exemple plus bas). Voici la nouvelle procédure:

$$\mathit{Dec} \quad \frac{S = S_1 + (f(s_1, \ldots, s_n) \stackrel{?}{=} f(t_1, \ldots, t_n)))}{T; S \ \rightarrow \ T; (S_1 + \{(s_1 \stackrel{?}{=} t_1), \ldots, (s_n \stackrel{?}{=} t_n)\})}$$

$$\mathit{Prop} \quad \frac{S = S_1 + (x \stackrel{?}{=} t) \quad (x \notin V(t))}{T; S \ \rightarrow \ ((x \stackrel{?}{=} t) :: T); [t/x](S_1)}$$

$$\text{(A)} \quad \frac{S = S_1 + (t \stackrel{?}{=} x)}{T; S \ \rightarrow \ T; (S_1 + (x \stackrel{?}{=} t))} \qquad\qquad \text{(B)} \quad \frac{S = S_1 + (x \stackrel{?}{=} x)}{T; S \ \rightarrow \ T; S_1}$$

$$\frac{S = S_1 + (f(s_1, \ldots, s_n) \stackrel{?}{=} g(t_1, \ldots, t_m)) \quad (f \neq g)}{T; S \ \rightarrow \ \mathit{FAIL}}$$

$$\frac{S = S_1 + (x \stackrel{?}{=} t) \quad (x \in V(t))}{T; S \ \rightarrow \ \mathit{FAIL}}$$

La configuration initiale pour un problème $(t_1 \stackrel{?}{=} t_2)$ est $\mathit{nil}; \{(t_1 \stackrel{?}{=} t_2)\}$ (*nil* désigne la liste vide). Un état final est de la forme $T; \emptyset$. et la substitution $T^\dagger$ représentée par T est définie comme suit:

$$\mathit{nil}^\dagger = \mathit{id} \qquad\qquad ((x \stackrel{?}{=} t) :: T)^\dagger = [t/x] \circ T^\dagger \ .$$

Sur l'exemple de la question (B4), on obtient l'exécution suivante, en choisissant de traiter d'abord x_n, puis $x_{n-1}, \ldots$:

$$\begin{array}{l}
\mathit{nil} \ ; \ \{(x_1 \stackrel{?}{=} f(x_0, x_0)), \ldots, (x_n \stackrel{?}{=} f(x_{n-1}, x_{n-1}))\} \\
\quad \rightarrow \ (x_n \stackrel{?}{=} f(x_{n-1}, x_{n-1})) \ ; \ \{(x_1 \stackrel{?}{=} f(x_0, x_0)), , \ldots, (x_{n-1} \stackrel{?}{=} f(x_{n-2}, x_{n-2}))\} \\
\quad \vdots \\
\quad \rightarrow \ ((x_1 \stackrel{?}{=} f(x_0, x_0)) :: (\cdots (x_n \stackrel{?}{=} f(x_{n-1}, x_{n-1})))) \ ; \ \emptyset
\end{array}$$

On note que x_n n'apparaît pas dans $\{(x_1 \stackrel{?}{=} f(x_0, x_0)), \ldots, (x_{n-1} \stackrel{?}{=} f(x_{n-2}, x_{n-2}))\}$, et ainsi de suite pour $x_{n-1}, \ldots, x_1$. C'est donc à chaque fois le cas particulier suivant, économique, de la règle *Prop* qui peut être utilisé:

$$Prop\star \quad \frac{S = S_1 + (x \stackrel{?}{=} t) \quad (x \notin V(S_1) \cup V(t))}{T; S \;\rightarrow\; ((x \stackrel{?}{=} t) :: T); S_1}$$

Si par contre on exécute la nouvelle procédure en traitant d'abord x_1, puis $x_2, \ldots$, on ne gagne rien, et on termine avec une liste de taille exponentielle:

$$(\cdots((x_2 \stackrel{?}{=} f(f(x_0, x_0), f(x_0, x_0))) :: (x_1 \stackrel{?}{=} f(x_0, x_0)))) \; .$$

(B6) La correction résulte des propriétés suivantes.

- On appelle solution d'un état $T; S$ toute substitution de la forme $\sigma \circ T^\dagger$, où σ est une solution de S. Alors, pour chaque réécriture $T; S \rightarrow T_1; S_1$, on peut montrer que $T; S$ et $T_1; S_1$ ont les mêmes solutions (détails omis). Ainsi, si $nil; S \rightarrow \cdots \rightarrow T; \emptyset$, alors l'ensemble des solutions de S est l'ensemble des substitutions de la forme $\sigma \circ T^\dagger$, où σ est quelconque (car toute substitution est solution de l'ensemble vide d'équations). On a donc non seulement que $T^\dagger$ est une solution, mais que c'est la plus générale.

- Reste à montrer la terminaison de la procédure (quelque soit l'exécution). On utilise l'ordre lexicographique sur (n_1, n_2, n_3), où

  - n_1 est le nombre d'éléments de $V(S)$,
  - n_2 est le nombre de symboles fonctionnels apparaissant dans S,
  - n_3 est le nombre déquations de la forme $(t \stackrel{?}{=} x)$ dans S.

  On vérifie que chacune des règles fait décroître ce triplet. Vérifions le pour *Dec* et *Prop*: *Dec* laisse n_1 invariant et diminue n_2 de 2, tandis que *Prop* diminue n_1 de 1, car $x \in V(S)$ et $x \notin V([t/x](S_1))$. CQFD

On met le doigt pour terminer sur une seconde source d'inefficacité, qui reste une fois que l'on a systématisé l'usage de bonnes stratégies telles que celle utilisée dans la question (B5).

(C1) Soit S un ensemble d'équations, toutes de la forme $x = t$, et soit X l'ensemble des variables qui apparaissent dans S comme membre gauche d'une équation. Soient $x, y \in X$. On pose $x < y$ si S contient une équation $(y \stackrel{?}{=} t)$ telle que t n'est pas une variable et $x \in V(t)$. Montrer que si S a une solution, la clôture réflexive et transitive $<^\star$ de $<$ est une relation d'ordre, i.e., est anti-symétrique. Quelle stratégie pour l'unification cette propriété suggère-t-elle?

(C2) Quelle source d'inefficacité cette stratégie laisse-t-elle non traitée?

Preuve. (C1) Soit σ une solution de S. Si $x < y$, alors $\sigma(y) = \sigma(t)$ a $\sigma(x)$ comme sous-terme strict. Donc, si $x <^\star y$, alors $\sigma(x)$ est un sous-terme de $\sigma(y)$, strict si $x \neq y$. Il en résulte que $<^\star$ est anti-symétrique. Cela veut dire qu'il existe une variable x maximale pour $<$. En effet, sinon, on pourrait construire une suite infinie $x_1 < x_2 < \cdots < x_n < \cdots$, ce qui implique $x_i = x_j$ pour deux variables de cette suite (finitude de X), et contredit l'anti-symétrie.
Ce choix permet de s'approcher de la règle *Prop*$\star$ (cf. question (B5)). On peut toujours, à partir d'un état donné $T; S'$, se ramener par *Dec*, (A) et (B) à un état $T; S$ où S ne contient que des équations de la forme $(x \stackrel{?}{=} t)$ $(t \neq x)$. Puis on peut choisir x maximal comme défini ci-dessus et appliquer *Prop* à x:

$$\mathit{Prop}{\star\star}\ \frac{S = S_1 + (x \stackrel{?}{=} s)\ \ (x \text{ maximal}) \quad S_2 = \{(y \stackrel{?}{=} t) \in S \mid y \neq x \text{ et } t \notin V\}}{T; S \ \to\ ((x \stackrel{?}{=} s) :: T); (S_2 \cup [s/x](S_1 \setminus S_2))}$$

(C2) Dans $S_1 \setminus S_2$, on trouve des équations de la forme $x = s'$, qui deviennent $s = s'$, et des équations de la forme $y = x$, qui deviennent $y = s$. Considérons un problème $f(x, x, x) = f(t_1, t_2, t_3)$ où t_1, t_2, t_3 sont de gros termes. La procédure d'unification amène sur $\{(x \stackrel{?}{=} t_1), (x \stackrel{?}{=} t_2), (x \stackrel{?}{=} t_3)\}$, puis, par exemple, sur $\{(t_1 \stackrel{?}{=} t_2), (t_1 \stackrel{?}{=} t_3)\}$: on a dupliqué t_1, ce qui implique de pratiquer séparément la décomposition sur les deux équations $(t_1 \stackrel{?}{=} t_2)$ et $(t_1 \stackrel{?}{=} t_3)$: ceci est une source d'inefficacité, car t_1 ca être exploré deux fois. Une façon de traiter ce problème est de remplacer les équations $(x \stackrel{?}{=} s)$ par des *multi-équations* de la forme $(X \stackrel{?}{=} M)$, où X est un ensemble de variables et M un multi-ensemble (ensemble dont les éléments peuvent être répétés) de termes qui ne sont pas des variables. L'exemple ci-dessus conduit à $(\{x\} \stackrel{?}{=} \{t_1, t_2, t_3\})$. Plus généralement, une éxécution de *Prop* $\star\star$ conduit à une multi-équation $(\{x, y, \ldots\} \stackrel{?}{=} \{s, s', \ldots\})$. On prend à droite un multi-ensemble plutôt qu'un ensemble parce qu'il peut bien se faire que par exemple $t_2 = t_3$, mais celà prend du temps de le savoir (et cette reconnaissance est d'ailleurs intégrée au processus d'unification). On peut écrire une troisième procédure adaptée de celle de la question (B5), basée sur les multi-équations (en formulant le problème initial $(t_1 \stackrel{?}{=} t_2)$ en $(\{x\} \stackrel{?}{=} \{t_1, t_2\})$ (x variable n'apparaissant ni dans t_1 ni dans t_2), et prouver que cette procédure est linéaire si l'on ne tient pas compte de fonctions annexes comme la recherche du maximum, et quasi-lináire si l'on tient compte de toutes les sous-procédures. Cette procédure est dûe à Martelli et Montanari. CQFD

10 Diagrammes 2-dimensionnels

Lorsqu'on écrit un terme $h(t_1, \ldots, t_m)$, il y a une dimension verticale (de h à un des t_i) et une dimension horizontale (entre deux t_i). La syntaxe qui suit prend ce point de vue 2-dimensionnel. On se donne une signature Σ d'opérateurs f, munis d'une arité constituée de deux entiers naturels: arité$(f) = (m, n)$. Comprendre que f a m entrées et n sorties. (Le symbole h ci-dessus est d'arité $(m, 1)$.) A partir de cette signature, on construit une syntaxe de diagrammes comme suit (chaque diagramme a un type $m \to n$):

$$\frac{\text{arité}(f) = (m,n)}{[i,f,j] : i+m+j \ \to\ i+n+j} \qquad \frac{}{id : m \ \to\ m} \qquad \frac{u : m \ \to\ n \quad v : n \ \to\ p}{v \circ u : m \ \to\ p}$$

Donc un diagramme u est de la forme $\cdots \circ [i, f, j] \circ \cdots$, et chaque $[i, f, j]$ est appelé un diagramme élémentaire. On représente $[i, f, j]$ par une figure comportant i fils verticaux à gauche, une boîte représentant f au centre avec m fils entrants et n fils sortants, puis j fils verticaux à droite. La composition consiste à placer deux tels réseaux l'un au-dessus de l'autre, en branchant les sorties du premier sur les entrées du second.

On note $i' + u + j' = \cdots \circ [i + i', f, j + j'] \circ \cdots$ (incrémentation sur chaque diagramme élémentaire; graphiquement, on rajoute i' fils à gauche et j' fils à droite); si $u : m \to n$, alors $i' + u + j' : i' + m + j' \to i' + n + j'$. On identifiera des diagrammes modulo des équations. On supposera toujours celle-ci:

$$\frac{\text{arité}(f) = (m,n) \quad \text{arité}(f') = (m',n') \quad i' \geq (i+n)}{[i',f',j'] \circ [i,f,j] = [i,f,j-m'+n'] \circ [i'+m-n,f',j']}$$

La condition $i' \geq (i + n)$ garantit que f' n'utilise aucun des fils de sortie de f, et l'équation exprime une propriété dite d'isotopie: on peut faire bouger f et f' verticalement, sans altérer le sens du diagramme. Il pourra être pratique (et non ambigü grâce à l'équation ci-dessus) d'écrire ce diagramme $i + f + k + f' + j'$, où $k = i' - i - n = j - j' - m'$. On omettra i si $i = 0$ (idem pour k, j'). Plus généralement on peut définir, pour tous u, v, $i + u + j + v + k$ tel que:

$$\frac{u : m \ \to\ n \quad v : p \ \to\ q}{i+u+j+v+k : i+m+j+p+k \ \to\ i+n+j+q+k}$$

Pour distinguer la coïncidence de l'égalité modulo des équations, on écrira $u \equiv v$ dans le premier cas (ceci incluant cependant l'identification ci-dessus) et $u = v$ dans le second.

On va s'intéresser d'abord à la signature $\Sigma = \{\mu, \eta\}$, avec arité$(\mu) = (2,1)$ et arité$(\eta) = (0,1)$. On dessine μ comme un trangle équilatéral pointant vers le bas (avec deux fils entrants sur les sommets du haut et un fil sortant du sommet du bas), et η comme une petit cercle sans fil entrant et avec un seul fil sortant. On interprète les diagrammes $u : m \to n$ sur cette signature comme des fonctions $\overline{u} : \{1, \ldots, m\} \to \{1, \ldots, n\}$, comme suit: $\overline{u \circ v} = \overline{u} \circ \overline{v}$ et

$$\overline{[i, \mu, j]}(n) = \begin{cases} n & \text{si } n \leq (i+1) \\ n-1 & \text{si } n \geq (i+2) \end{cases} \qquad \overline{[i, \eta, j]}(n) = \begin{cases} n & \text{si } n \leq i \\ n+1 & \text{si } n > i \end{cases}$$

(A1) Montrer que cette interprétation est une surjection sur l'ensemble des fonctions monotones (de $\{1, \ldots, m\}$ dans $\{1, \ldots, n\}$, pour tous m, n).

(A2) Donner un ensemble E d'équations tel que pour tous u, v, on a $u = v$ modulo E ssi $\overline{u} = \overline{v}$ (i.e., sont la même fonction)..

Preuve. (A1) L'interprétation d'un diagramme élémentaire est monotone, et la composition de deux fonctions monotones est monotone. On montre ensuite que toute fonction monotone $\omega : \{1, \ldots, m\} \to \{1, \ldots, n\}$ est interprétation d'un diagramme. On remarque d'abord qu'on a $f = \alpha_{p_1} + \cdots + \alpha_{p_n}$, où p_i est le cardinal de $f^{-1}(i)$, où α_p est l'unique fonction de $\{1, \ldots, p\}$ dans $\{1\}$ et où $f + g$ est défini comme suit si $f : \{1, \ldots, q\} \to \{1, \ldots, r\}$: $(f+g)(n) = f(n)$ si $n \leq q$, et $(f+g)(n) = g(n-q) + r$ si $n > q$. D'autre part, on a $\overline{u+v} = \overline{u} + \overline{v}$. Donc le diagramme recherché est somme de diagrammes a_{p_i} tels que $\overline{a_{p_i}} = \alpha_{p_i}$, obtenus par combinaisons simples de μ et η:

$$a_0 = [0, \eta, 0] \qquad a_1 = id \qquad a_p = [0, \mu, 0] \circ (1 + a_{p-1}) \;\; (p \geq 2)\ .$$

(A2) On pose $f^\star = a_{p_0} + \cdots + a_{p_n}$. On prouve, pour tout u, que $u = (\overline{u})^\star$, De là, on conclut comme suit. si $\overline{u} = \overline{v}$, alors $u = (\overline{u})^\star \equiv (\overline{v})^\star = v$. On procède par récurrence sur le nombre de diagrammes élémentaires de u. Si $u = id$ ou si u est élémentaire, $(\overline{u})^\star \equiv u$. Si $u = [i, \eta, j] \circ v$, alors par récurrence $v = (\overline{v})^\star$ est égal à une somme de diagrammes a_p, et l'on a alors

$$u = (i + a_0 + j) \circ (\cdots + a_p + \cdots + a_q + \cdots) = (\cdots + a_p + \cdots + a_0 + \cdots + a_q + \cdots)\ .$$

Si $u = [i, \mu, j] \circ v$, alors un petit dessin convaint aisément qu'il s'agit de prouver:

$$(\dagger) \quad [0, \mu, 0] \circ (a_p + a_q) = a_{p+q}$$

Voici les trois équations nécessaires (qui sont clairement valides, i.e., si $u = v$ est prouvable par ces équations, alors $\overline{u} = \overline{v}$):

$$\begin{array}{ll} (A_1) & [0,\mu,0] \circ [0,\mu,1] = [0,\mu,0] \circ [1,\mu,0] \\ (A_2) & [0,\mu,0] \circ [0,\eta,1] = id \\ (A_3) & [0,\mu,0] \circ [1,\eta,0] = id \end{array}$$

Pour $p = 0$, (†) résulte de (A_2). L'équation (A_3) correspond au cas $p = 1$ et $q = 0$. Si $p = 1$ et $q \geq 1$, † n'est autre que la définition de a_{q+1}. Enfin, si $p \geq 2$, on procède par récurrence sur q en utilisant (A_1):

$$\begin{array}{rcll} [0,\mu,0] \circ (a_p + a_q) & = & [0,\mu,0] \circ (([0,\mu,0] \circ (1 + a_{p-1})) + a_q) & \\ & = & [0,\mu,0] \circ (([0,\mu,0] + 1) \circ (1 + a_{p-1} + a_q) & \\ & = & ([0,\mu,0] \circ [0,\mu,1]) \circ (1 + a_{p-1} + a_q) & \\ & = & ([0,\mu,0] \circ [1,\mu,0]) \circ (1 + a_{p-1} + a_q) & \text{(par } (A_1)) \\ & = & ([0,\mu,0] \circ (1 + ([0,\mu,0] \circ (a_{p-1} + a_q))) & \\ & = & [0,\mu,0] \circ (1 + a_{p+q-1}) & \text{(par récurrence)} \\ & = & a_{p+q} & \text{(par définition)} \end{array}$$

CQFD

On passe maintenant aux fonctions de domaine fini non nécessairement monotones.

(B1) Montrer que toute fonction peut s'écrire comme composition d'une fonction monotone et d'une permutation.

(B2) Quel opérateur faut-il pour représenter les permutations? Montrer comme en (A1) que toute permutation est interprëtation d'un diagramme écrit à l'aide de ce seul opérateur. On utilisera pour une permutation π sur $\{1,\ldots,n\}$ une décomposition de la forme $\pi = (\beta_p + q) \circ (1 + \pi')$, où π' est une permutation sur $\{1,\ldots,n-1\}$ obtenue à partir de la restriction de π à $\{2,\ldots,n\}$, et β_p est une permutation circulaire. Préciser en quel sens cette décomposition est unique.

(B3) Même question que (A2) en remplaçant la signature $\{\mu,\eta\}$ par celle réduite au nouvel opérateur introduit en (B2). Préciser l'usage de la propriété d'unicité évoquée ci-dessus.

Preuve. (B1) Soit $f : \{1,\ldots,m\} \to \{1,\ldots,n\}$. Soit π une permutation de $\{1,\ldots,m\}$ telle que $f^{-1}(1)$ apparaisse avant $f^{-1}(2)$, etc... dans la liste $\pi(1),\ldots,\pi(n)$. Alors $f \circ \pi$ est monotone, et f s'écrit $(f \circ \pi) \circ \pi^{-1}$.

(B2) Il est bien connu que toute permutation est un produit de transpositions. Il suffit donc d'introduire un seul nouvel opérateur $\tau : 2 \rightarrow 2$ avec pour interprétation:

$$\overline{[i,\tau,j]}(n) = \begin{cases} n & \text{si } n \leq i \\ i+2 & \text{si } n = i+1 \\ i+1 & \text{si } n = i+2 \\ n & \text{si } n > i+2 \end{cases}$$

On dessine cet opérateur à l'aide du symbole $\times$ (qui visualise l'échange des entrées et sorties). Il faut comme en (A1) définir un diagramme canonique $\pi^\star$ associé à une permutation $\pi : \{1,\ldots,n\} \rightarrow \{1,\ldots,n\}$, tel que $\overline{\pi^\star} = \pi$. On applique l'algorithme suivant: considérer d'abord la restriction de π à $\{2,\ldots,n\}$, puis procéder à l'insertion de $\pi(1)$ (cf. π' et β_p dans l'indication). Formellement, par renommage, la restriction $\pi : \{2,\ldots,n\} \rightarrow \{1,\ldots,\pi(1)-1,\pi(1)+1,\ldots,n\}$ devient $\pi' : \{1,\ldots,n-1\} \rightarrow \{1,\ldots,n-1\}$ définie par:

$$\pi'(i) = \begin{cases} \pi(i+1) & \text{si } \pi(i+1) < \pi(1) \\ \pi(i+1)-1 & \text{si } \pi(i+1) > \pi(1) \end{cases}$$

On définit $\beta_p : \{1,\ldots,p+1\} \rightarrow \{1,\ldots,p+1\}$ (permutation circulaire) par:

$$\beta_p(1) = p+1 \qquad \beta_p(i) = i-1 \ \ (i > 1)\ .$$

En prenant $p = \pi(1)-1$ et $q = n-p-1$, on obtient:

$$(\beta_p + q)(i) = \begin{cases} \pi(1) & \text{si } i = 1 \\ i-1 & \text{si } 1 > i \leq \pi(1) \\ i & \text{si } i > \pi(1) \end{cases}$$

$$(1+\pi')(i) = \begin{cases} 1 & \text{si } i = 1 \\ \pi(i)+1 & \text{si } \pi(i) < \pi(1) \\ \pi(i) & \text{si } \pi(i) > \pi(1) \end{cases}$$

ce qui donne bien π en composant. Il reste à exprimer β_p en fonction de τ (décomposition d'un cycle en produit de transpositions). On a $\beta_p = \overline{b_p}$, où b_p est comme suit:

$$b_0 = id \qquad b_p = (1 + b_{p-1}) \circ [0,\tau,p-1] \ \ (p \geq 1)\ .$$

On pose $\pi^\star = (b_p + q) \circ (1 + \pi'^\star)$ ($\pi'^\star$ est bien défini par récurrence sur n). Cette décomposition est unique en ce sens que s'il existe r, s, u tels que $\pi^\star =$

$(b_r + s) \circ (1 + u)$, alors $r = p, s = q$ et $\overline{u} = \pi'$. En particulier, si $u \equiv \pi''^{\star}$, alors $\pi'' = \pi'$.

(B3) Soit $u : n \to n$. On prouve $u = (\overline{u})^{\star}$ par récurrence sur n, puis sur la construction de u. Pour le cas de base, $u \equiv (\overline{u})^{\star}$ (noter que seuls des b_0 et un b_1 sont utilisés). Voici les règles qui sont utilisées pour le pas de récurrence:

$$\begin{array}{ll} (B_1) & [0,\tau,0] \circ [0,\tau,0] = id \\ (B_2) & [0,\tau,1] \circ [1,\tau,0] \circ [0,\tau,1] = [1,\tau,0] \circ [0,\tau,1] \circ [1,\tau,0] \end{array}$$

Soit $u = [i,\tau,j] \circ u'$ et $(\overline{u'})^{\star} = (b_p + q) \circ (1 + v')$. Il y a quatre cas:

- $i > p$: On a $u = [i,\tau,j] \circ (b_p + q) \circ (1 + v') = (b_p + q) \circ (1 + v'')$, où $v'' \equiv [i-1,\tau,j] \circ v'$. En appliquant la récurrence à $v'' : n-1 \to n-1$, et par la propriété d'unicité, on obtient $u = (b_p + q) \circ (1 + \overline{v''}^{\star}) \equiv \overline{u}^{\star}$.
- $i = p$: Alors $u = (b_{p+1} + (q-1)) \circ (1 + v') \equiv (\overline{u})^{\star}$.
- $i = p - 1$: En faisant la figure, on voit qu'en une application de (B_1) on arrive à $(b_{p-1} + (q+1)) \circ (1 + v') \equiv (\overline{u})^{\star}$.
- $i < p - 1$: En faisant la figure, on voit qu'un usage de (B_2) donne $u = (b_p + q) \circ (1 + v'')$, où $v'' \equiv [i,\tau,j-1] \circ v'$, et on conclut par récurrence (cf. premier cas).

CQFD

La décomposition (B1) et les propriétés (A1) et (B2) ont pour conséquence que la signature $\{\mu,\eta,\tau\}$ permet d'exprimer toute les fonctions de domaine fini et de codomaine fini. Cependant, cette décomposition n'étant pas unique, on ne peut pas simplement combiner (A1) et (B2) pour définir un diagramme associé à une fonction f quelconque (la non-canonicité peut cacher de nouvelles équations). Mais une décomposition directe de f inspirée de celle en (B2) fait l'affaire.

(C1) Soit $f : \{1,\ldots,n\} \to \{1,\ldots,n\}$. Trouver γ, q et f' (uniques) tels que $f = (\gamma + q) \circ (1 + f')$.

(C2) Même question que (A2) et (B3), cette fois avec $\{\mu,\eta,\tau\}$.

Preuve. (C1) On imite le développement fait en (B2): on considère d'abord la restriction de f à $\{2,\ldots,n\}$, puis on procède au placement de $f(1)$. On définit $f' : \{1,\ldots,n-1\} \to \{1,\ldots,n-1\}$ par:

$$f'(i) = \begin{cases} f(i+1) & \text{si } f(i+1) < f(1) \\ f(i+1) - 1 & \text{si } f(i+1) > f(1) \end{cases}$$

On définit $\gamma_p : \{1,\ldots,p+2\} \to \{1,\ldots,p+1\}$ par:

$$\gamma_p(1) = p+1 \qquad \gamma_p(i) = i-1 \ \ (i>1)$$

(noter que $\gamma_p(1) = \gamma_p(p+2)$). En prenant $p = \pi(1)-1$, $\gamma = \gamma_p$ et $q = n-p-2$, on obtient ce qui est demandé: $f = (\gamma+q)\circ(1+f')$. L'unicité de la décomposition est facile à vérifier.

(C2) La décomposition (C1) permet de définir une décomposition canonique, basée sur la représentation suivante de $\gamma_p = \overline{c_p}$:

$$c_0 = [0,\mu,0] \qquad c_p = (1+c_{p-1})\circ[0,\tau,p] \ \ (p \geq 1)\ .$$

On définit alors $f^\star = (c_p+q)\circ(1+(f')^\star)$. La preuve se déroule ensuite comme en (B3), mais il y a plus de cas. Voici les règles qui sont utilisées, en plus de celles déjà introduites.

$$\begin{array}{ll}
(C_1) & [0,\tau,0]\circ[0,\mu,1] = [1,\mu,0]\circ[0,\tau,1]\circ[1,\tau,0] \\
(C_2) & [0,\tau,0]\circ[1,\eta,0] = [0,\eta,1] \\
(C_3) & [0,\tau,0]\circ[1,\mu,0] = [0,\mu,1]\circ[1,\tau,0]\circ[0,\tau,1] \\
(D_1) & [0,\mu,0]\circ[0,\tau,0] = [0,\mu,0]
\end{array}$$

Soit $u = a\circ u'$, avec $(\overline{u'})^\star = (c_p+q)\circ(1+v')$. On distingue plusieurs cas selon la forme de a.

- $a = [i,\tau,j]$. On reprend seulement les cas qui changent par rapport à (B3).
  - $i = p$: On utilise (C_1).
  - $i = p-1$: On utilise (C_3) puis (B_1).
- $a = [i,\mu,j]$. Selon la valeur de i (en partant de $i = 0$), on fait usage de (C_3), ou de $[0,\mu,0]\circ[1,\mu,0]\circ[0,\tau,1] = [0,\mu,0]\circ[1,\mu,0]$ (elle-même conséquence de (D_1) et (A_1)), ou de (A_1).
- $a = [i,\eta,j]$. Si l'insertion de η vient perturber la configuration de c_p, on utilise (C_2) pour pousser η jusqu'à la droite du μ terminal de c_p, ce qui permet de cacher η dans la récurrence. On utilise ensuite (C_1), puis (B_2) autant que nécessaire pour reconstituer une configuration c_{p+1}.

  On peut noter que (C_3) est conséquence de (C_1) et (B_1) (faire deux (B_1)-expansions sur le membre gauche de (C_3), afin de faire apparaître comme sous-diagramme le membre droit de (C_1), appliquer (C_1) et terminer par une (B_1)-réduction).

(D1) Que faut-il ajouter pour représenter les relations et pas seulement les fonctions?

(D2) Que faut-il ajouter pour représenter les termes usuels "1-dimensionnels"? On rappelle que ces termes sont construits comme suit. Une variable x est un terme; si $t_1, \ldots, t_m$ sont des termes et si h est un symbole d'arité m, alors $h(t_1, \ldots, t_m)$ est un terme (en particulier, tout symbole d'arité 0 est un terme).

PREUVE. (D1) Il faut ajouter des opérateurs δ et ϵ avec arité$(\delta) = (1, 2)$ et arité$(\epsilon) = (1, 0)$.

(D2) On transforme l'arité "1-dimensionnelle" en arité 2-dimensionnelle. Ainsi, pour un symbole h d'arité m, on pose arité$(h) = (m, 1)$ (m entrées, une sortie). Pour représenter les termes, on utilise les duaux δ, ϵ et τ des opérateurs utilisés pour les fonctions finies. Ces opérateurs permettent respectivement de dupliquer, d'effacer et d'échanger des variables. Par exemple, si f et g sont d'arité 2, on construit $f(x, g(y, x))$ en dupliquant x (opérateur δ), puis en échangeant la seconde copie de x avec y (opérateur τ), puis en appliquant g aux sorties de cet échange, puis en appliquant f à la première copie de x et à la sortie de g. Au total, on a obtenu un diagramme dont les entrés sont nommées x (à gauche) et y (à droite), et qui a une seule sortie. L'opérateur d'effacement ϵ sert dès que l'on veut augmenter l'arité d'un terme: par exemple, si l'on veut faire apparaître $f(x, y)$ comme un terme d'arité 3, on peut le représenter par $f + \epsilon$, qui est un diagramme avec trois fils d'entrée, correspondant à x (à gauche), y (au milieu) et une autre variable (à droite). Formellement, pour tout n et pour tout choix de n variables distincles $x_1, \ldots x_n$ (qui nomment les fils entrants du diagramme) il y a une bijection entre les diagrammes d'arité $n \to 1$ et les termes f dont les variables sont parmi $x_1, \ldots x_n$.
Du point de vue équationnel, on prend les équations duales (i.e. obtenues en tournant les diagrammes de 180 degrés) de celles des questions (A2), (B3) et (C2), auxquelles on doit ajouter les lois qui permettent aux opérateurs h de la signature de traverser δ, ϵ et τ.
On termine par une remarque qui peut contribuer à éclairer le lien entre fonctions et termes. Si t est un terme dont les variables libres sont parmi $x_1, \ldots, x_n$, la liste (de longueur m) des occurrences de variables dans t, prises de gauche à

droite dans l'écriture linéaire du terme t, définit une fonction de $\{1, \ldots, m\}$ dans $\{1, \ldots, n\}$. Par exemple, la fonction f assoociée au terme $f(x, g(y, x))$ est la suivante: $f(1) = 1, f(2) = 2, f(3) = 1$.

CQFD

11 Algèbres combinatoires

Source. "Introduction to combinators and lambda-calculus" de R. Hindley, Cambridge University Press, 1986, pour les premières questions

Une algèbre combinatoire est une structure consistant en un ensemble X muni d'une loi de composition binaire $\bullet$ (que l'on ne suppose ni associative ni commutative), et de deux éléments distingués S et K vérifiant les équations suivantes, pour tous $x, y, z \in X$:

$$(K)\ \ (K \bullet x) \bullet y = x \qquad (S)\ \ ((S \bullet x) \bullet y) \bullet z = (x \bullet z) \bullet (y \bullet z)\ .$$

Il est commode de convenir de noter la loi $\bullet$ par simple juxtaposition, et d'associer les parenthèses vers la gauche, de sorte que les deux lois ci-dessus s'expriment plus sobrement comme suit: $Kxy = x$ et $Sxyz = (xz)(yz)$. Les éléments d'une algèbre combinatoire sont appelés des combinateurs.
La propriété fondamentale des algèbres combinatoires, qui est appelée la complétude combinatoire, fait l'objet de la question suivante. On considère les termes écrits avec des variables, K, S et $\bullet$. Par exemple, $S(xK)(SyzK)$ est un terme.

(A1) Montrer que dans toute algèbre combinatoire il existe un combinateur I tel que $Ix = x$ pour tout x.

(A2) Soit M un terme quelconque, dont les variables sont parmi $x, x_1, \ldots x_n$. Construire un terme M' (que l'on note $[x]M$) dont les variables sont parmi $x_1, \ldots x_n$ et tel que $([x]M)x = M$ est prouvable en utilisant les équations (S) et (K).

Preuve. (A1) On pose $I = (SK)K$, et on a bien $SKKx = Kx(Kx) = x$.

(A2) On définit $[x]M$ par récurrence sur la taille de M comme suit:

$$\begin{array}{lcll} [x]M & = & KM & \text{si } M = S, K \text{ ou } y \neq x \\ [x]x & = & I & \\ [x](M_1M_2) & = & S([x]M_1)([x]M_2) & \end{array}$$

Et on vérifie par récurrence aussi que $([x]M)x = M$. On se sert de l'équation (K) dans le premiers cas, de la question (A1) dans le second cas et de l'équation (S) dans le troisième cas. CQFD

On définit maintenant une notion d'algèbre combinatoire *affine*. Il s'agit d'une structure formée d'un ensemble L, d'une loi de composition binaire $\bullet$ (toujours notée par juxtaposition) et de quatre éléments distingués I, B, C, K satisfaisant les équations suivantes:

$$(I)\ \ Ix = x \quad (B)\ \ ((Bx)y)z = (xz)y \quad (C)\ \ ((Cx)y)z = x(yz) \quad (K)\ \ (Kx)y = x\ .$$

.

(B) Montrer que si L est une algèbre combinatoire affine possédant de plus un élément W tel que $(Wx)y = (xy)y$ pour tous x, y, alors L est une algèbre combinatoire.

Preuve. L'intuition est que W permet le codage de la duplication. Le combinateur S code lui aussi une duplication (celle de z dans $Sxyz$). Pour coder S, on remonte depuis le membre droit $(xz)(yz)$ de la règle (S). Un premier objectif consiste à écrire cette expression sous la forme Mzz, à l'aide de C et B (on souligne le sous-terme transformé à chaque étape):

$$\underline{(xz)(yz)} \quad = \quad \underline{C(xz)}yz \quad = \quad \underline{(CCx)zy}z \quad = \quad B(CCx)yzz\ .$$

On peut alors utiliser le combinateur W: $B(CCx)yzz \ = \ W(B(CCx)y)z$. Il reste à utiliser plusieurs fois la règle (C) pour obtenir une expression $Mxyz$, où M est un terme sans variable:

$$\begin{array}{rcl}
\underline{W(B(CCx)y)z} & = & \underline{CW(B(CCx))}yz \\
& = & \underline{C(CW)B(CCx)}yz \\
& = & C(C(CW)B)(CC)xyz\ .
\end{array}$$

Le combinateur $C(C(CW)B)(CC)$ a donc le comportement de S. Pour munir L d'une structure d'algèbre combinatoire, il suffit donc de choisir pour S le combinateur $C(C(CW)B)(CC)$. CQFD

On va maintenant construire une algèbre combinatoire particulière, définie par Abramsky. Soit T l'ensemble des suites u finies formées à partir des symboles L et R. On note ϵ la suite vide. Par exemple, $LLRLLRRRL \in T$. On considère l'ensemble des fonctions partielles de T dans T, noté $L_A = T \rightharpoonup T$ (A pour Abramsky) , que l'on va munir d'une opération binaire $\bullet$ comme suit. D'abord, on note L et R les fonctions de T dans T définies comme suit: $L(u) = Lu$ et $R(u) = Ru$, autrement dit L (R) prend une suite en entrée et rend en sortie

cette suite préfixée par L (R). On note que L et T sont injectives, et que les images $L(T)$ et $R(T)$ sont disjointes et que leur union n'est pas T tout entier, mais seulement T moins la suite vide: on a, précisément: $T = \{\epsilon\} \cup L(T) \cup R(T)$.
Soit $f, g \in L_A$. On définit une petite "machine" à l'aide de f et g, qui transforme à chaque étape un élément de T en un élément de T, précédé de ? ou de ! pour préciser si le calcul doit se poursuivre ou s'il est achevé:

$$\frac{f(u) = Rv \quad g(v) = w}{?u \ \rightarrow ?Rw} \qquad \frac{f(u) = Lv}{?u \ \rightarrow !v}$$

On pose $(f \bullet g)(u) = v$ si et seulement si il existe $v_1, \ldots, v_n$ ($n \geq 0$) tels que $?Lu \ \rightarrow ?v_1 \ \rightarrow \ \cdots \ \rightarrow ?v_n \ \rightarrow !v$. Informellement, on considère f comme une fonction partielle de $L(T) \cup R(T)$ dans $L(T) \cup R(T)$, que l'on peut représenter comme une boîte vers laquelle convergent un fil gauche et un fil droit et de laquelle divergent deux fils gauche et droit. Le fil divergeant à droite de la sortie de f est branché sur l'entrée de g (considéré comme une fonction de T dans T), et le fil de sortie de g est branché sur le fil convergeant à droite sur l'entrée de f. On forme ainsi une boucle entre f et g. Il peut être utile de porter des indications $+L$ et $+R$ sur les deux fils entrant sur f avant leur convergence, et $-L$ et $-R$ sur les deux fils sortant de f après leur divergence. Le flux d'information u rentre au départ par le fil convergeant à gauche sur l'entrée de f et devient donc Lu. Si $f(Lu)$ a la forme Lv, alors le calcul suit le fil gauche de sortie, en retirant le L, fournissant v comme résultat. Si $f(Lu)$ a la forme Rv, alors le calcul suit le fil droite de sortie, retire le R. C'est donc v qui est présenté à g, puis $Rg(v)$ à f. Il y a ensuite à nouveau deux cas sur $f(Rg(v))$,... . La boucle entre f et g est parcourue autant de fois qu'il y a d'étapes $\rightarrow$ dans l'exécution formelle de la machine, pour sortir, au bout de n boucles, $n \geq 0$ (ou ne jamais sortir, si $n = \infty$) par le fil divergeant à gauche de la sortie de f.
Noter que $f \bullet g$ est bien une fonction partielle: le calcul à partir de $?Lu$ peut bloquer, soit parce que f n'est pas défini, soit parce que g n'est pas défini, soit parce que le "dialogue" entre f et g que formalise la deuxième règle ci-dessus ne se termine pas (i.e., il existe $u_1, \ldots, u_n, \ldots$ tels que $?Lu \rightarrow ?u_1 \rightarrow \ldots \rightarrow ?u_n \rightarrow \ldots$).

(C1) Montrer que L_A est une algèbre combinatoire affine.

(C2) En vue d'obtenir une algèbre combinatoire, on modifie la définiton de T qui est maintenant l'ensembe engendré par la grammaire suivante:

$$u ::= \epsilon \mid Lu \mid Ru \mid \langle n, u \rangle \ \ (n \geq 0)\ .$$

On pose toujours $L_A = T \rightharpoonup T$, et on définit la fonction $! : L_A \rightarrow L_A$ comme suit: $(!f)(u)$ est défini si et seulement si $u = \langle n, v\rangle$ et alors $(!f)(u) = \langle n, f(v)\rangle$. L'idée sous-jacente à cette définition est que $!f$ représente un ensemble dénombrable de copies de f. On pose ensuite $f \bullet' g = f \bullet (!f)$. Montrer que L_A peut être munie d'une structure d'algèbre combinatoire dont la loi de composition binaire est $\bullet'$.

PREUVE. (C1) On montre successivement comment définir I, K, C et B.

- I est implémenté par un simple échange de L et R:

$$I(Lu) = Ru \qquad I(Ru) = Lu\ .$$

  Intuitivement, le réseau $I \bullet x$ fonctionne comme suit: $I \bullet x$ reçoit une entrée u, donc I reçoit l'entrée Lu, qui ressort de I sous la forme Ru, et est donc passée sous la forme u à x, qui repasse $R(x(u))$ à I qui ressort de I sous la forme $L(x(u))$, et ressort donc de $I \bullet x$ sous la forme $x(u)$. Donc $I \bullet x$ et x sont bien les mêmes fonctions partielles.

- K est une variation sur le même thème:

$$K(LLu) = Ru \qquad K(Ru) = LLu\ .$$

  Les redoublements de L tiennent compte de la plus grande complexité du câblage: on construit d'abord le réseau $K \bullet x$, puis le réseau $(K \bullet x) \bullet y$. On peut voir ce dernier comme un réseau $f \bullet y$, où f est une boîte noire dont le contenu se trouve être celui du réseau $K \bullet x$.

- Les définitions de B et C sont plus techniques. On demande à ces combinateurs de simuler un réseau:

$$\begin{array}{ll}
B(LLLu) = RLLu & C(LLLu) = RLu \\
B(LLRu) = RRu & C(RLu) = LLLu \\
B(LRu) = RLRu & C(RRu) = LRLu \\
B(RLLu) = LLLu & C(LRLu) = RRu \\
B(RLRu) = LRu & C(LRRu) = LLRu \\
B(RRu) = LLRu & C(LLRu) = LRRu
\end{array}$$

  Voici une explication des équations définissant B. Examinons le réseau $((B \bullet x) \bullet y) \bullet z$, qui doit simuler le réseau $(x \bullet z) \bullet y$. L'équation $B(LLLu) =$

$RLLu$ achemine u jusqu'à x, considéré comme placé au sein du réseau $(x \bullet z) \bullet y$: le R de $RLLu$ assure que LLu est acheminé sur x, et LLu assure la traversée de (la simulation de) $(x \bullet z) \bullet y$, puis de $x \bullet z$. L'équation $B(LLRu) = RRu$ ($B(LRu) = RLRu$) exprime que la sortie de z (y) doit être rebranchée sur l'entrée R (LR) de x. Les trois dernières équations gèrent la sortie de x. L'équation $B(RLLu) = LLLu$ achemine vers la sortie de $((B \bullet x) \bullet y) \bullet z$ une réponse sortant de $(x \bullet z) \bullet y$, d'où les trois L à droite et les deux L à gauche. L'équation $B(RLRu) = LRu$ ($B(RRu) = LLRu$) met en oeuvre la demande LRu (Ru) de branchement sur l'entrée de y (z).

Le code de C s'explique comme suit. On considère $((c \bullet x) \bullet y) \bullet z$. Pour commencer, C passe la main à x. Si x ne consulte pas son argument $y \bullet z$, i.e. si sa sortie est de la forme Lu, reçue par C avec le préfixe R, alors le calcul prend fin, avec la valeur u, préfixée par LLL. Si x rend un Ru, il faut passer u à $y \bullet z$, i.e., passer Lu à y, d'où le préfixe LR devant Lu. Si y rend Lu, il faut passer u à l'entrée droite de x (troisième ligne). Si y rend Ru, on passe la main à z. Enfin le résultat de z doit être passé à l'entrée droite de y.

(C2) Voici la définition de W:

$$\begin{array}{lcl}
W(LLu) & = & R\langle 0, LLu\rangle \\
W(R\langle 0, Lu\rangle) & = & LLu \\
W(R\langle 0, R\langle n, u\rangle\rangle) & = & LR\langle 2n, u\rangle \\
W(R\langle 0, LR\langle n, u\rangle\rangle) & = & LR\langle 2n+1, u\rangle \\
W(LR\langle 2n, u\rangle) & = & R\langle 0, R\langle n, u\rangle\rangle \\
W(LR\langle 2n+1, u\rangle) & = & R\langle 0, LR\langle n, u\rangle\rangle
\end{array}$$

L'idée de ce code est de séparer en deux l'espace des copies de y dans $(W \bullet (!x)) \bullet (!y)$ pour pouvoir distinguer celles de ces copies qui correspondent au premier et au deuxième $!y$ de $(x \bullet (!y)) \bullet (!y)$, respectivement. Une seule copie de x est nécessaire, d'où l'indice 0. Le code ressemble, hors cette gestion des copies, à celui de l'identité: W transmet son entrée la plus à gauche à x. Si x appelle son premier ou son deuxième argument, W transmet l'appel à $!y$, la parité des indices permettant ensuite de transmettre de manière appropriée à x le résultat de cet appel.

Il faut reprendre l'ensemble des combinateurs I, K, C, B dans ce nouveau cadre. L'idée est d'ignorer tout espace de duplication, et pour ce faire de ne jamais utiliser que l'indice 0. Encore faut-il insérer ces indices de manière appropriée (ce qui permet en définitive un meilleur “parsing” des équations). Voici les définitions revues:

$$
\begin{array}{ll}
I(Lu) = R\langle 0, u\rangle & K(LLu) = R\langle 0, u\rangle \\
I(R\langle 0, u\rangle) = Lu & K(R\langle 0, u\rangle) = LLu \\
\\
B(LLLu) = R\langle 0, LLu\rangle & C(LLLu) = R\langle 0, Lu\rangle \\
B(LLR\langle 0, u\rangle) = R\langle 0, Ru\rangle & C(R\langle 0, Lu\rangle) = LLLu \\
B(LR\langle 0, u\rangle) = R\langle 0, LRu\rangle & C(R\langle 0, Ru\rangle) = LR\langle 0, Lu\rangle \\
B(R\langle 0, LLu\rangle) = LLLu & C(LR\langle 0, Lu\rangle) = R\langle 0, Ru\rangle \\
B(R\langle 0, LRu\rangle) = LR\langle 0, u\rangle & C(LR\langle 0, Ru\rangle) = LLR\langle 0, u\rangle \\
B(R\langle 0, Ru\rangle) = LLR\langle 0, u\rangle & C(LLR\langle 0, u\rangle) = LR\langle 0, Ru\rangle
\end{array}
$$

On se trouve à présent en situation de pouvoir appliquer (B) pour conclure. CQFD

12 Sémantique des variables locales

On considère un mini-langage impératif, donné par la syntaxe suivante (X est une variable, n un entier, et op parcourt l'ensemble des applications usuelles d'addition, multiplication,...):

$$\begin{array}{ll} \text{Expressions} & e ::= !X \mid n \mid e \; op \; e \\ \text{Commandes} & c ::= (X := e) \mid c; c \mid new\ X.c \end{array}$$

Les deux premières commandes sont l'affectation et le séquencement. La dernière est une déclaration de variable locale. C'est une abréviation pour une syntaxe plus traditionnelle:

$$new\ X.c \equiv begin\ varX\ X := 0\ ;\ c\ end$$

On dit que c est le corps du bloc $new\ X.c$. La variable locale X n'est en vigueur que durant l'exécution de c. L'expression $!X$ désigne la valeur actuelle de la variable X (qu'on appelle aussi déréférenciation de X).

(A1) Quelles sont les valeurs de X et Y à la fin de l'exécution de

$$X := 7; Y :=!X; newX.(X :=!X+1; Y :=!Y+!X); X :=!X-!Y; Y :=!Y+1$$

Même question, en faisant l'hypothèse que Y est un alias de (ou un autre nom pour) X.

(A2) On dit qu'une occurrence d'une variable X ou d'une expression $!X$ est liée dans une commande c si X est locale dans un bloc $new\ X.c'$ de c, et qu'elle est libre si elle n'est pas liée. Indiquer les occurrences libres et liées de X et Y dans la commande de la question précédente. Formaliser en définissant par récurrence sur la construction des expressions (commandes) l'ensemble $VL(e)$ ($VL(c)$) des variables libres de (i.e. ayant des occurrences libres dans) e (c).

Preuve. (A1) La variable Y est mise à 7, puis à 8 (car la variable locale X en vigueur dans le bloc est mise à 1), puis à 9. La variable X est mise à 7 – valeur qui est restaurée à la suite de l'exécution du bloc – puis elle est mise à -1 (car Y vaut 8 à la sortie du bloc). Cette exécution fait l'hypothèse implicite que X et Y sont des variables différentes. Si elles sont deux noms pour la même variable (ou le même registre), alors les choses se passent de la même manière jusqu'à la sortie du bloc, à ceci près que la mise à 8 de Y affecte aussi la variable globale X, donc les affectations suivantes mettent X (et donc Y) à 0, puis Y (et donc X) à 1.

(A2) Les occurrences libres (liées) sont surlignées (soulignées) ci-dessous:

$$\overline{X} := 7; \overline{Y} := \overline{!X}; \mathit{new}\, X.(\underline{X} := \underline{!X}+1; \overline{Y} := \overline{!Y}+\underline{!X}); \overline{X} := \overline{!X}-\overline{!Y}; \overline{Y} := \overline{!Y}+1$$

Voici la définition des ensembles VL:

$$\begin{array}{ll} \mathit{VL}(!X) = X & \mathit{VL}(X := e) = \{X\} \cup \mathit{VL}(e) \\ \mathit{VL}(e_1\ \mathit{op}\ e_2) = \mathit{VL}(e_1) \cup \mathit{VL}(e_2) & \mathit{VL}(c_1; c_2) = \mathit{VL}(c_1) \cup \mathit{VL}(c_2) \\ & \mathit{VL}(\mathit{new}\ X.c) = \mathit{VL}(c) \setminus \{X\} \end{array}$$

CQFD

Pour décrire mathématiquement la signification ou *sémantique* des expressions et des commandes, on se donne un ensemble L d'adresses (*locations* en anglais), On note V l'ensemble des valeurs que prennent les expressions (les entiers par exemple). Un environnement ρ est une fonction (partielle) associant une adresse à chaque variable dans son domaine de définition. Un état s est une fonction de L dans $V \cup \{\mathit{unused}\}$, où *unused* est une valeur spéciale supposée ne pas appartenir à V. On note S l'ensemble des états.

(B1) Définir par récurrence sur la formation des expressions (commandes) l'interprétation de e dans un environnement ρ dont le domaine de définition contient $\mathit{VL}(e)$ et dans un état s tel que

$$\forall\, l \in L\ \ (\exists\, X \in \mathit{VL}(e)\ \ l = \rho(X) \Rightarrow s(l) \neq \mathit{unused})\ .$$

On note l'interprétation $[\![e]\!]\rho s$, et on doit avoir $[\![e]\!]\rho s \in V$. Définir semblablement l'interprétation $[\![c]\!]\rho s$ d'une commande, telle que $[\![c]\!]\rho s \in S$.

Que peut-on dire de $([\![c]\!]\rho s)(l)$ si $(\not\exists X \in \mathit{VL}(c)\ \ l = \rho(X))$? Donner une condition suffisante sur ρ_1, s_1 et ρ_2, s_2 pour que $[\![e]\!]\rho_1 s_1 = [\![e]\!]\rho_2 s_2$ et pour qu'un énoncé "du même genre" soit vérifië pour les commandes. Puis donner une condition suffisante pour que $[\![c]\!]\rho_1 s = [\![c]\!]\rho_2 s$ pour tout s. Que dire en particulier d'une commande c telle que $\mathit{VL}(c) = \emptyset$?

(B2) Quelle est l'interprétation de la commande de la question (A1)?

(B3) Partant d'une commande c telle que $VL(c) = \emptyset$, la définition de $[\![c]\!]$ dans l'environnement vide et dans état constant *unused* appelle récursivement des définitions de sous-commandes $[\![c']\!]\rho s$. Quelles propriétés ces couples (ρ, s) satisfont-ils? On fera l'hypothèse que tous les X apparaissant dans des sous-commandes de la forme $new\ X.c$ sont distincts. Cette hypothèse est-elle restrictive? Comment peut-on rendre non-ambigü le fait que l'on ne souhaite pas que X et Y soient l'un alias de l'autre dans l'exemple de la question (A1)?

(B4) Montrer que si $X \notin VL(c)$, alors $[\![new\ X.c]\!]\rho s = [\![c]\!]\rho s$.

PREUVE. (B1) Si f est une fonction, on note $f[v \leftarrow w]$ la fonction f' égale à f partout sauf éventuellement en v, et telle que $f'(v) = w$.

$$\begin{array}{l}
[\![!X]\!]\rho s = s(\rho(X)) \\
[\![n]\!]\rho s = n \\
[\![e_1\ op\ e_2]\!]\rho s = ([\![e_1]\!]\rho s)\ op\ ([\![e_2]\!]\rho s)
\end{array}$$

$$\begin{array}{l}
[\![X := e]\!]\rho s = s[\rho(X) \leftarrow [\![e]\!]\rho s] \\
[\![c_1; c_2]\!]\rho s = [\![c_2]\!]\rho([\![c_1]\!]\rho s) \\
[\![new\ X.c]\!]\rho s = ([\![c]\!]\rho[X \leftarrow l]\ s[l \leftarrow 0])\ [l \leftarrow unused] \quad (s(l) = unused)
\end{array}$$

Dans la dernière ligne, on utilise une adresse quelconque telle que $s(l) = unused$. Il faut donc vérifier que l'interprétation ne dépend pas de ce choix. Pour cela, on montre plusieurs propriétés par récurrence (et l'on répond ainsi aux autres questions posées), *en même temps* que l'on définit $[\![e]\!]$ et $[\![c]\!]$. Si A est un ensemble de variables, on pose $\rho_1 \sim_A \rho_2$ si $\forall X, Y \in A\ (\rho_1(X) = \rho_1(Y) \Leftrightarrow \rho_2(X) = \rho_2(Y))$. Voici les propriétés (pour tous $\rho, s, \rho_1, s_1, \rho_2, s_2$):

$$\begin{array}{ll}
(1) & (\nexists X \in VL(c)\ \ l = \rho(X)) \\
 & \quad \Rightarrow ([\![c]\!]\rho s)(l) = s(l) \\[1em]
(2e) & (\forall X \in VL(e)\ \ s_1(\rho_1(X)) = s_2(\rho_2(X))) \\
 & \quad \Rightarrow [\![e]\!]\rho_1 s_1 = [\![e]\!]\rho_2 s_2 \\[1em]
(2c) & (\rho_1 \sim_{VL(c)} \rho_2 \text{ et } \forall X \in VL(c)\ \ s_1(\rho_1(X)) = s_2(\rho_2(X))) \\
 & \quad \Rightarrow (\forall X \in VL(c)\ \ ([\![c]\!]\rho_1 s_1)(\rho_1(X)) = ([\![c]\!]\rho_2 s_2)(\rho_2(X)))
\end{array}$$

Examinons quelques cas. L'hypothèse de $(2e)$ appliquée à X donne $(2e)$ pour $!X$. Pour le cas $X := e$, posons $\epsilon = [\![e]\!]\rho_1 s_1 = [\![e]\!]\rho_2 s_2$ (égalité acquise par récurrence),

$s'_1 = s_1[\rho_1(X) \leftarrow \epsilon]$ et $s'_2 = s_2[\rho_2(X) \leftarrow \epsilon]$. On a donc $s'_1(\rho_1(X)) = s'_2(\rho_2(X))$. Soit $Y \in VL(e)$, différent de X. Si $\rho_1(X) \neq \rho_1(Y)$, alors $\rho_2(X) \neq \rho_2(Y)$ par la première hypothèse de $(2c)$, donc $s'_1(\rho_1(Y)) = s_1(\rho_1(Y)) = s_2(\rho_2(Y)) = s'_2(\rho_2(Y))$. Si $\rho_1(X) = \rho_1(Y)$, alors $\rho_2(X) = \rho_2(Y)$, donc $s'_1(\rho_1(Y)) = \epsilon = s'_2(\rho_2(Y))$. La conclusion de $(2c)$ est ainsi vérifiée dans tous les cas. Pour $c_1; c_2$, on applique $(2c)$ deux fois, en prenant la seconde fois la première conclusion pour seconde hypothèse.

On peut maintenant montrer que l'interprétation de *new* $X.c$ est bien définie (et le même genre d'argument permet détablir $(2c)$ pour *new* $X.c$). Supposons que $l_1 \neq l_2$ sont telles que $s(l_1) = s(l_2) = \mathit{unused}$. Posons $\rho_1 = \rho[X \leftarrow l_1]$, $s_1 = s[l_1 \leftarrow 0]$, $\rho_2 = \rho[X \leftarrow l_2]$ et $s_2 = s[l_2 \leftarrow 0]$. On va montrer que ρ_1, s_1, ρ_2, s_2 vérifient les hypothèses de $(2c)$ (pour c). Montrons d'abord que $\rho_1 \sim_{VL(c)} \rho_2$. Soient $Y, Z \in VL(c)$ tels que $\rho_1(Y) = \rho_1(Z)$. Si Y et Z sont différents de X, ou si $Y = Z = X$, il n'y a rien à prouver. Montrons que ce sont les seuls cas possibles. Le seul cas qui reste est, disons $Z = X$ et $Y \neq X$. On a par construction d'une part $s(\rho_1(Y)) = s(\rho(Y)) \neq \mathit{unused}$ puisque $Y \in VL(\mathit{new}\ X.c)$, et d'autre part $s(\rho_1(X)) = s(l_1) = \mathit{unused}$; donc $\rho_1(Z) \neq \rho_1(Y)$. On passe maintenant à la seconde hypothèse. On a par construction $s_1(\rho_1(X)) = s_2(\rho_2(X))$. Pour $Y \neq X$, on note que $\rho(Y) \neq l_1, l_2$, car sinon on aurait $s(\rho(Y)) = \mathit{unused}$, contrairement à l'hypothèse faite sur ρ et s. On a donc

$$s_1(\rho_1(Y)) = s_1(\rho(Y)) = s(\rho(Y)) = s_2(\rho_2(Y)) \ .$$

Soient A_1 et A_2 les deux membres droits de la définition de $[\![\mathit{new}\ X.c]\!]\rho s$, écrits en utilisant l_1 et l_2 respectivement. On doit montrer $A_1(l) = A_2(l)$ pour tout l. On distingue plusieurs cas:

- $l \neq l_1, l_2$ et $\exists\, Y \in VL(\mathit{new}\ X.c)\ \ l = \rho(Y)$. Alors on a $A_1(l) = A_1(\rho(Y)) = A_1(\rho_1(Y)) = A_2(\rho_2(Y)) = A_2(l)$ par $(2c)$.

- $l = l_1$. Alors $A_1(l_1) = \mathit{unused}$ par construction, tandis que $([\![c]\!]\rho_2 s_2)(l_1) = s_2(l_1) = s(l_1) = \mathit{unused}$ par (1), d'où $A_2(l_1) = \mathit{unused}$. Le cas $l = l_2$ est traité de même.

- $l \neq l_1, l_2$ et $\nexists\, Y \in VL(\mathit{new}\ X.c)\ \ l = \rho(Y)$. Alors, en utilisant (1), on a $A_1(l) = s(l) = A_2(l)$.

L'interprétation est donc correctement définie. On montre maintenant les propriétés restantes. Suppossons que $\forall\, X \in VL(c)\ \ \rho_1(X) = \rho_2(X)$. Alors, pour tout s, la seconde condition de $(2c)$ est vérifiée en prenant $s_1 = s = s_2$, et l'on obtient donc que $([\![c]\!]\rho_1 s)(l) = ([\![c]\!]\rho_2 s)(l)$, pour tout l image d'une variable libre de c. Pour les autres adresses, on utilise (1): $([\![c]\!]\rho_1 s)(l) = s(l) = ([\![c]\!]\rho_2 s)(l)$. On a donc $[\![c]\!]\rho_1 s = [\![c]\!]\rho_2 s$. En résumé, l'interprétation $[\![c]\!]\rho$ est déterminée par la restriction de ρ aux variables libres de c. Si $VL(c) = \emptyset$, alors $[\![c]\!]\rho_1 = [\![c]\!]\rho_2$ pour tous ρ_1, ρ_2. On peut donc omettre l'environnement et écrire simplement $[\![c]\!]$. Tout ceci est aussi vrai pour les expressions.

(B2) Il y a deux cas (c désigne ici la commande de la question (A1))

- $\rho(X) = l_1 \neq l_2 = \rho(Y)$. Alors $[\![c]\!]\rho s = s[l_1 \leftarrow -1, l_2 \leftarrow 9]$.
- $\rho(X) = l = \rho(Y)$. Alors $[\![c]\!]\rho s = s[l \leftarrow 1]$.

(B3) Si l'on part d'une commande sans variables libres et d'un environnement vide, l'environnement ne s'élargit que lors d'entrées dans des blocs $new\, X.c$ (sous-commandes de la commande de départ). On montre, par récurrence sur la taille du sous-programme, que ($dom(\rho)$ désigne le domaine de définition de ρ):

$(P0)$ $X \in dom(\rho) \Rightarrow X$ apparaît dans une sous-commande $new\ X.c$
$(P1)$ $X_1, X_2 \in dom(\rho)$ et $X_1 \neq X_2 \Rightarrow \rho(X_1) \neq \rho(X_2)$
$(P2)$ $X \in dom(\rho) \Leftrightarrow s(\rho(X)) \neq unused$

Les propriétés $(P0)$ et $(P1)$ sont conséquences immédiates de l'hypothèse faite. Cette hypothèse n'est pas restrictive, car on peut toujours renommer une variable liée sans changer l'interprétation (détails fastidieux, mais faciles). Soient ρ, s l'environnement et l'état à l'entrée de $new\ X.c$, et soient $\rho_1 = \rho[X \leftarrow l]$ et $s_1 = s[l \leftarrow 0]$. On a $\rho_1(s_1(X)) = 0 \neq unused$, donc, en utilisant la récurrence sur ρ, s, on a $(P2)(\Rightarrow)$ pour ρ_1, s_1. Il faut aussi regarder les autres cas de commandes. Si ρ, s sont l'environnement et l'état à l'entrée de $c_1; c_2$, alors ρ et $[\![c_1]\!]\rho s$ sont l'environnement et l'état à l'entrée de c_2. et l'on conclut grâce à la propriété suivante (pour tous c, ρ, s):

$$s(l) \neq unused \Rightarrow ([\![c]\!]\rho s)(l) \neq unused$$

Cette propriété peut se montrer par récurrence avec les autres propriétés énoncées dans la solution de la question (B1). La vérification (facile) de $(P2)(\Leftarrow)$ est laissée au lecteur.
Pour garantir que Y n'est pas alias de X (ou réciproquement), il suffit d'encapsuler la commande c de la question (A1) dans $c' = new\ X.(new\ Y.c)$. "Officiellement", $[\![c']\!]$ est une commande "boîte noire" qui s'exécute avec l'environnement

vide et n'a aucun effet sur l'état. Mais la définition par récurrence de $[\![c]\!]$ à partir de $[\![c']\!]$ garantit que deux adresses différentes seront attribuées à X et à Y.

(B4) Sous l'hypothèse $X \notin VL(c)$, la définition de $[\![new\ X.c]\!]\rho s$ se simplifie:

$$\begin{aligned}
[\![new\ X.c]\!]\rho s &=_{\text{cf. (2c)}} ([\![c]\!]\rho s)\,[l \leftarrow unused] \\
&=_{\text{cf. (1)}} [\![c]\!]\rho s
\end{aligned}$$

CQFD

Le cadre adopté ci-dessus exige beaucoup de soins dans la formulation de propriétés invariantes de l'interprétation, propriétés qui jouent un rôle essentiel pour permettre de vérifier par exemple l'équivalence de la question (B4). Dans la seconde partie de ce sujet, on adopte une approche plus abstraite et plus élégante, qui évite le recours explicite à un ensemble d'adresses. Au lieu de considérer un seul ensemble S d'états, on considère une famille d'ensembles W d'états, également appelés *mondes*, qui sont de la forme V^n (V est comme ci-dessus l'ensemble des valeurs, et V^n est l'ensemble des n-uplets d'éléments de V). L'interprétation est définie dans chaque monde. On considère trois types d'objets: les expressions sont de type *int*, les commandes sont de type *comm*, et les variables sont de type *var*. On a ($X \to Y$ désigne l'ensemble des fonctions de X dans Y):

$$\begin{array}{ll}
[\![e]\!]W\rho \in [\![int]\!]W & [\![int]\!]W = W \to V \\
[\![c]\!]W\rho \in [\![comm]\!]W & [\![comm]\!]W = W \to W \\
\rho(X) \in [\![var]\!]W & [\![var]\!]W = (V \to [\![comm]\!]W) \times [\![int]\!]W
\end{array}$$

Comme ci-dessus, on fait l'hypothèse que le domaine de définition de ρ contient l'ensemble des variables libres.

(C1) Ecrire les fonctions d'interprétation des expressions et des commandes dans ce nouvau cadre.

(C2) Reprendre la question (B4) dans ce cadre.

(C3) Quelle idée se trouve-t-elle derrière la définition de $[\![var]\!]W$?

PREUVE. (C1) Voici les nouvelles égalités sémantiques:

$$
\begin{aligned}
&[\![!X]\!]W\rho s = \pi'(\rho(X))(s)\\
&[\![n]\!]W\rho s = n\\
&[\![e_1\ op\ e_2]\!]W\rho s = [\![e_1]\!]W\rho s\ op\ [\![e_2]\!]W\rho s\\
\\
&[\![X := e]\!]W\rho s = \pi(\rho(X))([\![e]\!]W\rho s)(s)\\
&[\![c_1; c_2 W]\!]\rho s = [\![c_2]\!]W\rho([\![c_1]\!]W\rho s)\\
&[\![new\ X.c]\!]W\rho s = \pi([\![c]\!](W \times V)\overline{\rho}[X \leftarrow (a, e)]\ (s, 0))
\end{aligned}
$$

Et voici les définitions des symboles utilisés dans ces équations:

- $\pi : X \times Y \to X$ et $\pi' : X \times Y \to Y$ sont les projections: $\pi(x, y) = x$ et $\pi'(x, y) = y$.
- $\overline{\rho}(X) = (\overline{a}_X, \overline{e}_X) \in [\![var]\!](W \times V)$ est défini si et seulement si $\rho(X) = (a_X, e_X) \in [\![var]\!]W$ est défini, et

$$\overline{a}_X(v')(s, v) = (a_X(v')(s), v) \qquad \overline{e}_X(s, v) = e(s)\ .$$

- a, e sont définis par:

$$a(v')(s, v) = (s, v') \qquad e(s, v) = v\ .$$

(C2) C'est la conséquence de la propriété suivante, que l'on peut montrer aisément par récurrence sur les commandes: si $W' = W \times V^i$ et si π_i est la fonction de projection de W' sur W, alors, pour tous $c, \rho, s, v_1, \ldots, v_i$,

$$\pi_i([\![c]\!]W'\overline{\rho}(s, v_1, \ldots, v_i)) = [\![c]\!]W\rho s$$

où $\overline{\rho}$ est défini comme ci-dessus.

(C3) La définition modélise les deux usages d'une variable: lecture ($!X$) et écriture ($X := e$). Ces deux usages sont respectivement pris en compte dans la deuxième et dans la première composante d'un élément de $[\![var]\!]$. La première composante est une fonction, qui prend en argument la valeur à écrire. CQFD