Documents de travail
pour l'oral d'informatique des ENS

Pierre-Louis Curien et Yves Robert

Septembre 1999

ENS Cachan   ENS Lyon   ENS Ulm

Préface

Voici rassemblés quelques documents qui nous ont servi de base de travail pour faire passer l'épreuve orale d'Informatique commune aux trois ENS Cachan, Lyon, et Ulm en 1999. Cette épreuve consiste en une interrogation au tableau d'une durée de 45 minutes, sans préparation préalable et sans partie "pratique" (programmation sur machine).

Pour permettre à tous les candidats de mieux préparer l'épreuve, nous avons décidé de diffuser largement le présent document. Le fichier postcript est disponible sur les pages Web des auteurs.

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-mêmes 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-être 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 !

Septembre 1999
Pierre-Louis Curien, `Pierre-Louis.Curien@ens.fr`,
`http ://www.dmi.ens.fr/~curien`
Yves Robert, `Yves.Robert@ens-lyon.fr`,
`http ://www.ens-lyon.fr/~yrobert`

Table des matières

1 Réductions de 3-SAT

Source. D. Kozen, “The Design and Analysis of Algorithms”, Springer Verlag (1992), pour les parties (B), (C) et (D).

Le problème 3-SAT est la satisfiabilité de formules conjonction de clauses qui sont chacune la disjonction de trois littéraux. Pour le problème NAE (“not all equal”), on se donne la contrainte supplémentaire que les trois littéraux ne doivent pas recevoir la même valeur, i.e., il doit y en avoir au moins un vrai (pour la validité de la clause) et au moins un faux. Pour le problème OIT (“one in three”) , on se donne (par rapport à 3-SAT) la contrainte que l'un seulement des littéraux doit être vrai. Chacun de ces problèmes P est paramétré par le nombre n de variables dont on dispose pour écrire les formules, et par le nombre p de clauses (notation $P(n,p)$).

(A) Montrer que NAE et OIT ne sont pas plus faciles que 3-SAT.

Preuve. (NAE) A chaque clause $c = a_1 \vee a_2 \vee a_3$ d'une formule, on associe deux clauses

$$c' = a_1 \vee a_2 \vee x \qquad c'' = \overline{x} \vee a_3 \vee F,$$

où x est une variable fraîche. On montre que c est satisfiable si et seulement si $c' \wedge c''$ est NAE satisfiable avec les mêmes valeurs pour a_1, a_2, a_3 (ce dernier point est pour le recollage de toutes les clauses). Supposons d'abord c satisfaite. Si $a_3 = V$, alors c'' est NAE satisfaite tandis que c' est NAE satisfaite si a_1 et a_2 ont une valeur différente ; si a_1 et a_2 ont la même valeur, il suffit de prendre l'autre valeur de vérité pour x. Si $a_3 = F$, il n'y a pas de choix pour x, qui doit être F, et c' est alors NAE satisfaite, car la satisfaction de c implique que a_1 ou a_2 est vrai.
Réciproquement, supposons que c' et c'' sont NAE satisfaites, et supposons (pour obtenir une contradiction) que $a_1 = a_2 = a_3 = F$. Ceci impose $x = V$, mais alors c'' est une disjonction de F. Contradiction. Nous avons ainsi réduit 3-SAT à NAE, et plus précisément 3-SAT(n,p) à NAE$(n+p, 2p)$, soit une complexité de traduction polynomiale. (N.B. : ce codage est une variante de la technique de résolution employée aussi dans la question (B).)

(OIT) A chaque clause $c = a_1 \vee a_2 \vee a_3$, on associe trois clauses

$$c' = \overline{a_1} \vee x \vee y \quad c'' = y \vee a_2 \vee z \quad c''' = z \vee t \vee \overline{a_3},$$

où x, y, z et t sont des variables fraîches. On montre que c est satisfiable si et seulement si $c' \wedge c'' \wedge c'''$ est OIT satisfiable. Supposons d'abord c satisfaite. Si $a_2 = V$, alors la contrainte OIT impose $y = z = F$, et on satisfait c' et c'' en prenant respectivement $x = a_1$ et $t = a_3$. Si $a_2 = F$, alors la contrainte OIT impose $y = \overline{z}$. Supposons aussi que $a_3 = F$ (et donc $a_1 = V$). Alors $z = t = F$ (clause c'''), donc $y = V$ (clause c''), et c' peut être satisfaite avec $x = F$. La situation $a_2 = F, a_1 = F$ (et donc $a_3 = V$) est symétrique (prendre $x = y = F$, $z = V$ et $t = F$). Il reste le cas $a_2 = F$ avec $a_1 = a_3 = V$: les contraintes sont $x = \overline{y}$, $y = \overline{z}$ et $z = \overline{t}$. On peut donc fixer x et définir successivement $y = \overline{x}$, $z = x$ et $t = \overline{x}$.
Réciproquement, supposons que c', c'' et c''' sont OIT satisfaites, et que $a_1 = a_2 = a_3 = F$. Ceci impose $x = y = F$ (clause c') et $z = t = F$ (clause c''') : contradiction, car alors c'' est insatisfiable. On a donc réduit 3-SAT(n, p) à OIT$(n+4p, 3p)$. CQFD

(B) Montrer que 3-SAT n'est pas plus facile que SAT.

PREUVE. Il faut coder une clause $c = a_1 \vee \cdots \vee a_m$ par un ensemble de clauses de trois littéraux chacune. Ceci peut se faire à l'aide de $m - 3$ variables fraîches $x_1, \ldots, x_{m-3}$ et de $m - 2$ clauses, comme suit :

$$\begin{array}{lcll} c_1 & = & a_1 \vee a_2 \vee x_1 & \\ c_i & = & \overline{x_{i-1}} \vee a_{i+1} \vee x_i & \quad (1 < i < m-2) \\ c_{m-2} & = & \overline{x_{m-3}} \vee a_{m-1} \vee a_m & \end{array}$$

La clef de ce codage est la forme suivante de la règle de résolution : pour toutes formules A, B, C et toute variable propositionnelle n'apparaissant ni dans A ni dans B ni dans C, alors $(x \vee A) \wedge (\overline{x} \vee B) \wedge C$ est satisfiable si et seulement si $(A \vee B) \wedge C$ est satisfiable. C'est facile à vérifier avec des tables de vérité. Cela peut aider de voir x comme un marqueur. Si $A \vee B$ est vrai parce que A (B) est vrai, on marque $x = F$ ($x = V$). Alors on a, de proche en proche :

$$\begin{array}{ccc} c_1 \wedge c_2 \wedge c_3 \wedge \cdots \wedge c_{m-2} & (a_1 \vee a_2 \vee a_3 \vee x_2) \wedge c_3 \wedge \cdots \wedge c_{m-2} & \\ \cdots & (a_1 \vee \cdots \vee a_{m-2} \vee x_{m-3}) \wedge c_{m-2} & c\,. \end{array}$$

Il est facile de voir que cette traduction est linéaire en la taille de la formule : chaque clause de m littéraux est traduite en $m - 2$ clauses de trois littéraux.
CQFD

(C) Réduire 2-SAT à un problème mettant en jeu les composantes (fortement) connexes d'un graphe orienté (idée : les littéraux sont les sommets, chaque clause $a_1 \vee a_2$ donne lieu à deux arêtes).

PREUVE. Avec l'indication donnée, on s'oriente assez naturellement vers le connecteur implication (transitif comme les chemins d'un graphe) : les deux arêtes dans lesquelles se traduit la clause $a_1 \vee a_2$ sont $(\overline{a_1}, a_2)$ et $(\overline{a_2}, a_1)$. En procédant ainsi pour toutes les clauses d'une formule B (conjonction de clauses de cette forme), on construit ainsi un graphe G_B qui a pour sommets les littéraux de B et leurs négations.. On montre la propriété suivante.

(P) La formule B est insatisfiable si et seulement si il existe un littéral a sommet de G_B tel qu'il existe un chemin de a à $\overline{a}$ *et* de $\overline{a}$ à a.

On rappelle d'abord la notion de composante fortement connexe. C'est une classe d'équivalence de la relation $a \cong b$ si et seulement si il existe un chemin de a à b et un chemin de b à a. On montre d'abord que si v est une valuation pour laquelle B est satisfaite, alors v doit être constant sur chaque composante fortement connexe de G_B. Soit C une composante, $a \in C$ un sommet fixé de C, et a' un sommet quelconque de C. Supposons d'abord que $v(a) = V$, et soit $a = a_1, \ldots, a_n = a'$ un chemin de a à a'. Par définition du graphe, les clauses $(\overline{a_i}, a_{i+1})$ (plus suggestives sous la forme $a_i \Rightarrow a_{i+1}$) sont des clauses de B. Donc $v(B) = V$ implique $v(a_i \Rightarrow a_{i+1}) = V$, ce qui de proche en proche implique $v(a_i) = V$ pour tout i ; en particulier $v(a') = V$, donc v vaut V sur toute la composante. Si $v(a) = F$, supposons que $v(a') = V$, alors on déduit comme ci-dessus en suivant un chemin de a' à a que $v(a) = V$: contradiction. Donc v vaut F sur toute la composante.
On peut maintenant prouver la propriété (P). Supposons d'abord que G_B a un sommet a tel que dans l'énoncé et que pourtant B est satisfiable pour une valuation v. On vient de voir que v doit être constant sur chaque composante fortement connexe. Or a et $\overline{a}$ sont dans la même composante, ce qui implique $v(a) = v(\overline{a})$: contradiction.
Réciproquement, supposons que G_B n'a pas deux littéraux opposés dans la même composante. Il faut construire une valuation telle que $v(B) = V$. Les composantes fortement connexes établissent une partition des littéraux de B, chaque classe de cette partition devant recevoir la même valeur de vérité. Reste à choisir pour chaque classe cette valeur commune. On considère le graphe orienté dont les sommets sont les composantes, deux composantes C_1 et C_2 étant reliées si et seulement si il existe une arête (a_1, a_2) de G_B telle que $a_1 \in C_1$ et $a_2 \in C_2$. Ce graphe est acyclique (sinon, en fixant un cycle et une arête (C_1, C_2) de ce cycle, de représentant (a_1, a_2), on pourrait construire en suivant le cycle un chemin de a_2 à a_1 dans G_B, ce qui contredit l'appartenance de a_1 et a_2 à des composantes différentes). On peut donc faire un tri topologique des composantes (i.e., une énumération L des composantes telle que pour tous C_1, C_2, si (C_1, C_2) est une

arête, alors C_2 apparaît après C_1 dans l'énumération). On peut réaliser un tel tri en temps linéaire. On peut aussi déterminer le graphe des composantes en temps linéaire. (Ces deux points peuvent faire l'objet d'un autre sujet...). On peut maintenant construire $v(x)$ pour chaque variable x. Soient C_1 et C_2 les composantes (distinctes par hypothèse) contenant x et $\overline{x}$, respectivement. On pose $v(x) = F$ si C_1 apparaît avant C_2 dans L et $v(x) = V$ dans le cas contraire. Reste à prouver que ces choix sont cohérents. Supposons qu'une clause $a_1 \vee a_2$ ne soit pas satisfaite, i.e., que $v(a_1) = v(a_2) = F$. On distingue deux cas.

- a_1 et a_2 appartiennent à la même composante C_1. Soit C_2 la composante de $\overline{a_1}$. Puisque $(\overline{a_1}, a_2)$ est une arête de G_B, (C_2, C_1) est une arête, et par conséquent C_2 doit apparaître avant C_1 dans L. Or $\overline{a_1} \in C_2$ et $a_1 \in C_1$, et d'après la définition de v, ceci implique $v(\overline{a_1}) = F$: contradiction.
- a_1 et a_2 appartiennent à deux composantes distinctes C_1 et C_2, respectivement. Soient C_1' et C_2' les composantes de $\overline{a_1}$ et $\overline{a_2}$. Les positions de ces composantes dans la liste L sont comme suit :

  C_1 apparaît avant C_1' (définition de v)
  C_1' apparaît avant C_2 (tri topologique)
  C_2 apparaît avant C_2' (définition de v)
  C_2' apparaît avant C_1 (tri topologique)

  On a donc à la fois que C_1 précède C_2' et que C_2' précède C_1 (strictement) : contradiction.

Il reste à se convaincre que le critère (P) permet de tester la satisfiabilité en en temps linéaire, et aussi de construire un témoin v en un temps linéaire. Il suffit de passer en revue les éléments des composantes, une par une. On marque au fur et à mesure les variables x par le nom des composantes qui contiennent x et $\overline{x}$ (on peut utiliser un record dont les noms de champ sont les noms de variables). L'algorithme échoue s'il doit marquer la seconde fois x avec le même nom de composante que la première. Pour fabriquer v, c'est la même procédure, sauf que l'on marque F (V) sur x à la première visite si cette visite se fait du fait de la présence de x ($\overline{x}$) dans la composante courante. CQFD

On termine par une restriction de nature différente, qui conduit cependant à une analyse de complexité semblable. On va appeler SAT_n le problème SAT restreint aux formules qui n'ont pas plus de n occurrences de la même variable (x ou $\overline{x}$).

(D1) Montrer que SAT_3 n'est pas plus facile que SAT (et donc pas plus facile que SAT_n).

(D2) Montrer que SAT_2 est polynomial.

Preuve. (D1) Soit A une formule. Pour chaque x apparaissant dans A, on se donne m variables fraîches $x_1, \ldots, x_m$, où m est le nombre d'occurrences de x ou $\overline{x}$ dans A. On construit $A'' = A' \wedge c_x \wedge c_y \wedge \cdots$, où A' est obtenu en remplaçant la i-ème occurrence de la variable x par x_i (et ce pour toutes les variables de A) et

$$c_x = (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3) \wedge \cdots \wedge (\overline{x_m} \vee x_1)\ .$$

et de même pour les autres variables apparaissant dans A. Une façon plus agréable de lire c_x est $(x_1 \Rightarrow x_2 \Rightarrow \cdots \Rightarrow x_n \Rightarrow x_1)$. Il est clair que si A'' est satisfaite, alors tous les x_i ont la même valeur, et que A'' est satisfiable si et seulement si A l'est en prenant pour x cette valeur commune (et ce pour tout x). Il reste à remarquer que x_i apparaît exactement trois fois dans A'' (une fois dans A' et deux fois dans c_x). Il est facile de voir que la taille de A'' est linéaire en la taille de A : les clauses supplémentaires ont une taille $3m$, et la sommation des m est $0(\text{taille}(A))$.

(D2) On procède par réécriture de formules. Il est commode de voir une formule A comme un ensemble de clauses plutôt qu'une conjonction de clauses, et de voir chaque clause aussi comme un ensemble de littéraux. On écrit aussi $x \in A$ au lieu de $(\exists\, c \in A \;\; x \in c)$.

$$\frac{x \in A \quad \overline{x} \notin A}{A \;\rightarrow\; A \setminus \{c \in A \mid x \in c\}} \qquad \frac{\overline{x} \in A \quad x \notin A}{A \;\rightarrow\; A \setminus \{c \in A \mid \overline{x} \in c\}} \qquad \frac{x, \overline{x} \in c \in A}{A \;\rightarrow\; A \setminus \{c\}}$$

$$\frac{x \in c_1 \quad \overline{x} \in c_2 \quad c_1 \neq c_2 \quad c_1 \cup c_2 \neq \{x, \overline{x}\}}{A \;\rightarrow\; A \cup \{(c_1 \setminus \{x\}) \cup (c_2 \setminus \{\overline{x}\})\} \setminus \{c_1, c_2\}} \qquad \frac{c_1 = \{x\} \quad c_2 = \{\overline{x}\}}{A \;\rightarrow\; \text{FAIL}}$$

Il est facile de voir que si $A \rightarrow A'$ alors A est satisfiable si et seulement si A' est satisfiable, que si $A \rightarrow^* \text{FAIL}$ alors A est insatisfiable, et que si $A \rightarrow^* \emptyset$ alors A est satisfiable. Il est aussi facile de voir que la réécriture termine (à chaque étape une variable disparaît, et aucune nouvelle variable n'apparaît), et ne peut terminer que sur FAIL ou sur $\emptyset$. Le temps de l'algorithme est n fois le coût d'une étape (n est comme depuis le début le nombre de variables), qui est au pire linéaire en la taille de la formule courante, et a fortiori de taille(A). CQFD

2 Arbres de poids minimal

Source. Th. Cormen, C Leiserson et R. Rivest, “Introduction à l’Algorithmique”, Dunod (1994), chapitre 37, pour les questions (D) et (E).

On représente ici un graphe dont les arêtes sont étiquetées par un poids par une matrice symétrique (n, n) (n est le nombre de sommets du graphe) dont les entrées sont notées $d(i, j)$; si $d(i, j)$ est fini, alors il y a entre i et j une arête de i à j de poids $d(i, j)$, et on note l’absence d’arête entre deux sommets i et j par $d(i, j) = \infty$. On recherche un arbre (sous-graphe acyclique et connexe) couvrant (tous les n sommets sont sommets du sous-graphe) dont la somme des poids des arêtes soit minimale. Un tel arbre est appelé arbre de poids minimal (abréviation APM).

(A) Montrer l’existence d’au moins un APM, pour tout graphe G connexe muni d’une fonction de poids sur ses arêtes.

PREUVE. Il suffit de montrer qu’il existe un arbre couvrant : l’existence d’un APM résulte alors de la finitude. Il suffit d’enlever une par une des arêtes contribuant à un cycle. La connexité est un invariant pour ces transformations successives. Lorsque l’on ne peut plus enlever d’arêtes, c’est que le graphe est devenu acyclique. CQFD

(B1) Spécifier formellement l’algorithme suivant, dit algorithme de Prim : à chaque étape, ajouter un sommet et une arête dont le poids est minimal parmi toutes les arêtes menant d’un point non encore inclus vers un point inclus dans le sous-graphe.

(B2) Prouver la correction de l’algorithme.

(B3) Analyser grossièrement la complexité.

PREUVE. (B1) On utilise deux tableaux L et D de longueur n. L stocke des sommets ou _, et D stocke des distances ou _ (le symbole _ veut dire “déjà traité”). L’initialisation est (i_0 est un sommet choisi arbitrairement) :

$$\begin{array}{ll} i \neq i_0 \Rightarrow L[i] = i_0 & L[i_0] = \_ \\ i \neq i_0 \Rightarrow D[i] = d(i, i_0) & D[i_0] = \_ \end{array}$$

Voici une description formelle de l’algorithme :

$$\frac{D[j] = \min\{D[i] \mid D[i] \neq \_\}}{(D, L) \xrightarrow{!(L(j),j)} (D[\_/j], L[\_/j], j, 1)} \qquad \frac{}{(D, L, j, n) \longrightarrow (D, L)}$$

$$\frac{i < n \quad D[i] \leq d(i,j)}{(D, L, j, i) \longrightarrow (D, L, j, i+1)} \qquad \frac{i < n \quad D[i] > d(i,j)}{(D, L, j, i) \longrightarrow (D[d(i,j)/i], L[j/i], j, i+1)}$$

L'arbre couvrant est obtenu par effet de bord : Dans la première règle, $\xrightarrow{!(j_1,j_2)}$ indique l'output de l'arête (j_1, j_2). Les autres règles décrivent le travail de mise à jour des deux tableaux (nouvelle distance minimale et nouveau sommet le plus proche, pour chaque entrée non encore traitée). L'algorithme se termine lorsque $\{i \mid D[i] \neq \_\}$ est vide. La liste des output successifs indique alors une énumération de toutes les arêtes de l'APM.

(B2) On montre que chaque arbre partiel construit au cours de l'algorithme est inclus dans un arbre couvrant minimal (qu'on appellera APM courant). A la fin de l'algorithme, l'arbre partiel a n sommets, et coïncide donc avec tout APM le contenant. Soit (j_1, j_2) une arête ajoutée lors d'une étape $\xrightarrow{!(j_1,j_2)}$, et soient A_1 et A'_1 l'arbre partiel courant et l'APM courant avant cette étape, respectivement. Si $(j_1, j_2) \in A'_1$, alors A'_1 peut rester l'APM courant. Si $(j_1, j_2) \notin A'_1$, on exploite le fait que tout ajout d'une arête entre deux sommets d'un arbre crée un cycle. Or j_1 et j_2 sont bien des sommets de A'_1 puisque A'_1 est un arbre couvrant. En suivant ce cycle dans A'_1 en partant de j_1, soit j'_1 le dernier sommet rencontré qui est dans A_1 mais dont le successeur j'_2 n'est pas dans A_1 (un tel sommet existe puisque j_2 n'est pas dans A_1 par construction). L'algorithme est tel qu'il garantit $d(j_1, j_2) \leq d(j'_1, j'_2)$. Soit $A'_2 = A'_1 \cup \{j_1, j_2\} \setminus \{j'_1, j'_2\}$. Il s'agit d'un arbre couvrant (même nombre de sommets et d'arêtes que A'_1), dont le poids est $\leq$ au poids de A'_1, donc égal au poids de A'_1 puisque A'_1 est minimal. A'_2 est donc aussi minimal, et on peut le choisir comme nouvel APM courant.

(B3) L'algorithme comprend $n - 1$ phases d'ajout d'une arête, entrecoupées d'étapes de mise à jour des tableaux. Il y a a priori n étapes de ce type pour chaque phase, d'où un coût quadratique en le nombre de sommets. On peut améliorer la situation si l'on utilise un tas ordonné D au lieu d'un tableau, et arriver à un algorithme en $O(p \log(n)$ (p étant le nombre d'arêtes), mais ce serait un autre sujet. CQFD

(C) Spécifier et prouver la correction de l'algorithme suivant, dit algorithme de Kruskal : à chaque étape, rajouter une arête dont le poids est minimal parmi toutes celles qui ne créent pas de cycle.

Preuve. On observe d'abord que ce n'est pas un arbre qui est progressivement construit comme dans l'algorithme de Prim, mais une forêt (i.e. un graphe acyclique, mais non nécessairement connexe). On restera évasif sur les structures de données utilisées. Il faut maintenir l'information sur l'appartenance des sommets de l'arbre courant aux composantes connexes de l'arbre courant. On suppose donc que l'on dispose d'une fonction cpst qui envoie les sommets sur les composantes. On suppose que l'on a rangé au départ les arêtes de G dans une pile, en empilant d'abord celles dont les poids sont les plus forts. Initialement, la fonction cpst est la fonction identité (il y a autant de composantes que de sommets). Pour réajuster la fonction cpst, on utilise la fonction suivante $\textsc{unir}_{C_1,C_2} : \{C_1, C_2, \ldots, C_n\} \rightarrow \{C_2, \ldots, C_n\}$:

$$\textsc{unir}_{C_1,C_2}(C) = \begin{cases} C & \text{si } C \neq C_1 \\ C_2 & \text{si } C = C_1 \end{cases}$$

(ainsi les composantes sont toujours nommées par des sommets). L'algorithme de Kruskal est alors ainsi spécifié :

$$\frac{P = (u,v) :: P' \quad \textsc{cpst}(u) = \textsc{cpst}(v)}{(\textsc{cpst}, P) \longrightarrow (\textsc{cpst}, P')}$$

$$\frac{P = (u,v) :: P' \quad \textsc{cpst}(u) \neq \textsc{cpst}(v)}{(\textsc{cpst}, P) \xrightarrow{!(u,v)} (\textsc{unir}_{\textsc{cpst}(u),\textsc{cpst}(v)} \circ \textsc{cpst}, P')}$$

L'algorithme s'arrête quand la pile est vide. La preuve de correction de l'algorithme est, mutatis mutandis, la même qu'en (B2). Il faut cependant noter un petit point subtil : on a besoin de la correction pour être sûr que l'algorithme termine sur un arbre couvrant, et non sur une forêt dont le nombre de sommets est $< n$. En effet, la preuve de correction assure non seulement que le choix d'une nouvelle arête (u,v) est sûr, mais aussi qu'il est possible : en effet, l'existence de l'APM courant implique qu'il y a bien dans la pile au moins une arête (u,v) qui ne crée pas de cycle. (On laisse de côté l'analyse de complexité qui implique une incursion dans les structures de données pour ensembles disjoints). CQFD

Remarque. Si les poids des arêtes du graphe sont tous distincts, l'APM est unique (voir la preuve de (B2), il n'y a pas de possibilité d'échange). Dans ce cas, les deux algorithmes construisent donc le même arbre, mais en choisissant les arêtes dans un ordre différent.

Le reste du sujet est consacré à l'application des APM à des approximations de facteur au plus 2 du problème du voyageur de commerce (TSP = traveling saleman's problem). Rappelons que ce problème consiste à trouver une *tournée* (i.e., un cycle qui visite tous les sommets) de poids minimum dans un graphe non orienté $G = (S, A)$ *complet* (i.e., $A = S \times S$) muni d'une fonction de poids d sur les arêtes. On fait l'hypothèse dite triangulaire :

$$(\Delta) \quad \forall\, u, v, w \in S \;\; d(u, v) \leq d(u, w) + d(w, v) \; .$$

Ce problème est connu pour être NP-complet, même sous l'hypothèse Δ. Le but de la question suivante est de construire deux algorithmes polynomiaux permettant de trouver une tournée de poids $\leq$ deux fois le poids d_{opt} d'un cycle optimal.

(D1) Construire une tournée de poids $\leq 2d_{opt}$ à partir d'un APM de G totalement construit.

(D2) Construire une tournée de poids $\leq 2d_{opt}$, en intégrant cette fois la construction de ce cycle dans l'algorithme de Prim.

PREUVE. (D1) Soit (S, A') un APM pour G. Considérons une tournée de poids d_{opt} comme un graphe. Il admet un arbre couvrant A'', qui est aussi un arbre couvrant de G (puisque tout sommet de G est visité par le cycle). On a donc

$$\begin{array}{ll} d(A') \leq d(A'') & \text{(minimalité de } A'\text{)} \\ d(A'') \leq d_{opt} & \text{(sous-graphe)} \end{array}$$

Il suffit donc de trouver un cycle H de poids $d(H) \leq 2d(A')$. (Il n'y a donc plus de référence au cycle optimal, mais seulement à un APM.). On va montrer que le H obtenu en énumèrant les sommets dans un ordre préfixe vis-à-vis de A' convient. Voici une façon de spécifier cet ordre. L'arbre A', comme tout arbre, peut s'écrire comme un terme avec la syntaxe suivante. On définit à la fois "A est un arbre" et "s est la racine de A" (notation $s = \textsc{root}(A)$).

- s est un arbre de racine s si $s \in S$.
- $s(A_1, \ldots, A_n)$ est un arbre de racine s si $A_1, \ldots, A_n$ sont des arbres disjoints (pas de sommets communs). (Les arêtes de $s(A_1, \ldots, A_n)$ sont celles des A_i plus les arêtes $(s, \textsc{root}(A_i))$.)

L'ordre préfixe $\textsc{op}$ est alors spécifié par

$$\textsc{op}(s) \;=\; s \qquad \textsc{op}(s(A_1,\ldots,A_n)) \;=\; (s\;\textsc{op}(A_1)\;\ldots\;\textsc{op}(A_n))\;.$$

On pose $H = (\textsc{op}(A')\;\textsc{root}(A'))$. La démonstration utilise un autre parcours de A', dit parcours complet, spécifié comme suit :

$$\textsc{pc}(s) \;=\; s \qquad \textsc{pc}(s(A_1,\ldots,A_n)) \;=\; (s\;\textsc{pc}(A_1)\;s\;\ldots\;s\;\textsc{pc}(A_n)\;s)\;.$$

Il est facile de voir par récurrence que chaque arête est parcourue exactement deux fois dans le parcours complet (une fois dans chaque sens). Par exemple, la portion $s\;\textsc{op}(A_i)\;s$, vue comme une suite d'arêtes, commence par $(s,\textsc{root}(A_i))$ et termine par $(\textsc{root}(A_i),s)$. Le rapport entre le parcours préfixe et le parcours complet apparaît plus évident si on les programme en les dérécursivant. On décore tous les sommets avec une étiquette notée en indice supérieur (initialement ^{1} partout). Voici une machine pour l'ordre préfixe :

$$\frac{A/u = s^1(A_1,\ldots,A_n)}{(A,u,\downarrow)\xrightarrow{!s}(A,u1,\downarrow)} \qquad \frac{A/u = s^1}{(A,u,\downarrow)\xrightarrow{!s}(A,u,\uparrow)}$$

$$\frac{A/u = s^n(A_1,\ldots,A_n) \quad u = vj}{(A,u,\uparrow)\longrightarrow(A,v,\uparrow)} \qquad \frac{A/u = s^i(A_1,\ldots,A_n) \quad i<n}{(A,u,\uparrow)\longrightarrow(A[{}^{i+1}/_{\circ}u],u(i+1),\downarrow)}$$

Et voici celle pour le parcours complet :

$$\frac{A/u = s^1(A_1,\ldots,A_n)}{(A,u,\downarrow)\xrightarrow{!s}(A,u1,\downarrow)} \qquad \frac{A/u = s^1}{(A,u,\downarrow)\longrightarrow(A,u,\uparrow)}$$

$$\frac{A/u = s^n(A_1,\ldots,A_n) \quad u = vj}{(A,u,\uparrow)\xrightarrow{!s}(A,v,\uparrow)} \qquad \frac{A/u = s^i(A_1,\ldots,A_n) \quad i<n}{(A,u,\uparrow)\longrightarrow(A[{}^{i+1}/_{\circ}u],u(i+1),\downarrow)}$$

Dans ces règles, on manipule des triplets dont la première composante est une variante décorée de A, la seconde composante codifie sous la forme d'un mot de nombres entiers naturels un pointeur sur un noeud de A, et la troisième est une marque qui indique si le pointeur monte ou desccend dans l'arbre. L'état initial est $(A,\epsilon,\downarrow)$ (où ϵ désigne le mot vide, lequel codifie la racine de A). La notation

$A[{}^{i+1}/_{\circ}u]$ signifie A dans lequel l'étiquette du noeud en u est maintenant ${}^{i+1}$. A chaque étape de la forme $\xrightarrow{!s}$, l'algorithme écrit le symbole s.
Dans les deux machines, les étiquettes en indice supérieur stockent l'information sur les branches déjà explorées. Les deux machines ont un contrôle identique et ne diffèrent que par leur output. Plus précisément, le parcours complet résulte des insertions suivantes (indépendantes les unes des autres) : si s_1 est une feuille suivie par le sommet s_2 dans le parcours préfixe, on insère entre les deux la suite des ancêtres successifs de s_1 (en commençant par le père s'_1) jusqu'au père de s_2. Il est alors facile (faire un dessin) de voir, en faisant un usage répété de (Δ), que $d(s_1, s_2) \leq d(s_1, s'_1) + d(s'_1, s'_2) + \cdots + d(s'_{n-1}, s'_l) + d(s'_l, s_2)$, d'où l'on déduit que le poids du parcours préfixe est moindre que le poids du parcours complet, lequel vaut deux fois le poids de l'APM.

(D2) L'invariant de la construction est le suivant. Si A_1 (de poids $\tilde{c}$) est l'arbre partiel courant, on a construit également un cycle pour les sommets de A_1 de poids c tel que $c \leq 2\tilde{c}$. Soit (u_1, v) la prochaine arête ajoutée à A_1. On modifie le cycle en insérant v entre u_1 et son successeur u_2 dans le cycle. Soient c' et $\tilde{c'}$ les nouveaux poids. On a :

$$\begin{array}{rcll} c' & = & c + d(u_1, v) + (d(v, u_2) - d(u_1, u_2)) & \\ & \leq & c + d(u_1, v) + d(u_1, v) & \text{(inégalité triangulaire)} \\ & \leq & 2(\tilde{c} + d(u_1, v)) & \text{(récurrence)} \\ & = & 2\tilde{c'} & \end{array}$$

(Cette construction admet une preuve de correction plus facile que la précédente, mais est dépendante de l'algorithme de construction de l'APM.) CQFD

Pour une mise en perspective, on regarde ce qui se passe sans l'hypothèse (Δ).

(E) Montrer qu'il n'existe pas d'approximation en temps polynomial de borne constante ρ pour le TSP, sauf si P=NP.

PREUVE. On réduit le problème de l'existence d'un cycle hamiltonien (cycle qui visite tous les sommets) dans un graphe $G = (S, A)$ à l'approximation polynomiale du TSP dans le graphe complet G' sur S, muni de la fonction de poids d suivante (n est le nombre de sommets) :

$$d(u, v) = \begin{cases} 1 & \text{si } (u, v) \in A \\ \rho n + 1 & \text{sinon} \end{cases}$$

(la fonction d ne vérifie pas (Δ)). La réduction résulte de la propriété suivante : l'approximation retourne un cycle hamiltonien de G si et seulement si il en existe un. Un sens est évident. Pour l'autre, supposons que G a un cycle hamiltonien. La définition de d implique que G admet un cycle hamiltonien si et seulement si G' admet une tournée de poids n (qui est alors minimale). Si une telle tournée existe, alors l'algorithme d'approximation retourne une tournée de poids $\leq \rho n$. Par définition de d, cette tournée ne peut emprunter aucun $(u, v) \notin A$. Elle se déroule donc entièrement dans G, i.e., elle fournit un cycle hamiltonien. CQFD

3 Programmation dynamique

Source. L'"Instructor's book" de Th. Cormen, C Leiserson et R. Rivest.

(A1) Spécifier un algorithme efficace pour une attribution optimale de m paires de skis de longueur $s_1, \ldots, s_m$, respectivement, à n skieurs ($m \geq n$) de taille $h_1, \ldots, h_n$, respectivement, via une fonction (injective) $f : \{1, \ldots, n\} \to \{1, \ldots, m\}$, f étant optimale lorsqu'elle minimise $\sum_{k=1}^{n} |s_{f(k)} - h_k|$. On donne l'indication suivante : soit $A[n, m]$ ce minimum. Définir $A[i, j]$ pour des valeurs plus petites $i \leq n$ et $j \leq m$ (lesquelles ?), par une équation de récurrence (i et j font référence aux i premiers skieurs et aux j premières paires de skis, respectivement).

(A2) Analyser la complexité (en veillant à affiner l'analyse de sorte à garantir que l'algorithme soit en $O(n \log n)$ si $m = n$).

(A3) Montrer qu'on peut avoir une meilleure complexité lorsque $n^2 = o(m)$. (Indication : se restreindre à $O(n^2)$ paires de skis).

Preuve. (A1) On considère le même problème que (A), mais en ne prenant que les i premiers skieurs et les j premières paires de skis (avec évidemment les mêmes longueurs et tailles). Pour écrire la récurrence définissant $A[i, j]$, on est naturellement amené à s'intéresser à la j-ème paire de skis et au i-ème skieur. Il y a deux cas. Si la solution optimale pour i, j n'utilise pas la j-ème paire de skis, alors $A[i, j] = A[i, j-1]$. Si la j-ème paire de skis est utilisée, il est tentant de l'affecter au i-ème joueur. On fait désormais l'hypothèse que les suites $s_1, \ldots, s_m$ et $h_1, \ldots, h_n$ sont rangées dans l'ordre croissant, ou plus précisément l'ont été dans une première phase de l'algorithme. On a la propriété suivante ;

(P) Si $f : \{1, \ldots, i\} \to \{1, \ldots, j\}$ (injective) est telle que $(\exists\, i_1 \leq i \;\; f(i_1) = j)$, alors $\sum_{k=1}^{i} |s_{f(k)} - h_k| \geq \sum_{k=1}^{i} |s_{f'(k)} - h_k|$, avec f' obtenue à partir de f en échangeant les images de i_1 et i (donc $f'(i_1) = f(i)$, $f'(i) = j$ et $f' = f$ ailleurs).

Preuve de (P). Intuitivement, la propriété (P) dit que si on a une attribution avec deux paires de ski pour deux skieurs donnés, mieux vaut (plus exactement on ne perd rien à) attribuer la plus petite paire au plus petit skieur et la plus grande paire au plus grand skieur. On pose

$$j_1 = f(i) \quad A = |s_j - h_{i_1}| + |s_{j_i} - h_i| \quad B = |s_{j_1} - h_{i_1}| + |s_j - h_i| \,.$$

Il faut montrer $A - B \geq 0$. On examine les différentes positions relatives de

s_{j_1}, s_j, h_{i_1} et h_i.

$$\begin{aligned} s_{j_1} \leq h_{i_1} \leq h_i \leq s_j : A - B &= (s_j - h_{i_1} + h_i - s_{j_1}) - (h_{i_1} - s_{j_1} + s_j - h_i) \\ &= 2(h_i - h_{i_1}) \ \geq \ 0 \end{aligned}$$

$$\begin{aligned} s_{j_1} \leq h_{i_1} \leq s_j \leq h_i : A - B &= (s_j - h_{i_1} + h_i - s_{j_1}) - (h_{i_1} - s_{j_1} - s_j + h_i) \\ &= 2(s_j - h_{i_1}) \ \geq \ 0 \end{aligned}$$

$$\begin{aligned} s_{j_1} \leq s_j \leq h_{i_1} \leq h_i : A - B &= (h_{i_1} - s_j + h_i - s_{j_1}) - (h_{i_1} - s_{j_1} + h_i - s_j) \\ &= 0 \end{aligned}$$

(On a omis les cas $h_{i_1} \leq s_{j_1} \leq h_i \leq s_j$ et $h_{i_1} \leq h_i \leq s_{j_1} \leq s_j$ qui sont similaires.)

Retour à la preuve de (A1). Par la propriété (P), on peut supposer que le i-ème skieur se voit attribuer la j-ème paire de skis. En effet, si ce n'est pas le cas pour f, alors f' tel que défini plus haut est meilleure que f, donc est optimale aussi, et attribue la j-ème paire de skis au i-ème skieur. Et on a $A[i,j] = A[i-1, j-1] + |s_j - h_i|$. Au total, on a prouvé :

$$A[i,j] = \min(A[i, j-1], (A[i-1, j-1] + |s_j - h_i|)) \ .$$

L'algorithme consiste donc à calculer ces $A[i,j]$, et si l'on veut calculer l'attribution f en même temps, on note $f(i) = j$ au passage si on est dans le deuxième cas.

(A2) Combien d'entrées $A[i,j]$ faut-il calculer ? Il est facile de voir que les appels récursifs depuis $A[n,m]$ laissent de côté les deux triangles gauche inférieur et droite supérieur de la matrice (n,m) qui ensemble ont une dimension (n,n). Donc, ce ne sont pas nm, mais $(m-n)n$ entrées qui sont à calculer. Si l'on compte le temps du tri des deux suites, on arrive donc à une complexité

$$O((m \log m) + (n \log n) + (m-n)n) \ .$$

Ceci est satisfaisant dans le cas particulier où $m = n$: on obtient alors $O(m \log m)$, et en effet, si $m = n$, il suffit de trier les deux suites et de poser $f = id$ (ce qui se voit par récurrence en utilisant la propriété (P)). (Noter que la complexité mal affinée $O(m \log m) + O(n \log n) + mn$ aurait donné $O(n^2)$.)

(A3) On remarque que si les listes sont triées, on peut se contenter, pour chaque skieur, de trouver la meilleure paire de skis et les $n-1$ meilleures paires de skis après la meilleure, de part et d'autre de la meilleure longueur. Cette fenêtre S_i de

taille $2n-1$ permet de prévoir les conflits avec les autres skieurs. Plus précisément, les paires de skis ainsi déterminées représentent un ensemble $S = \bigcup_{i=1}^{n} S_i$ de $n(2n-1)$ paires de skis avec de possibles répétitions tel que l'on peut définir une injection $f : \{1, \ldots, n\} \to S$ avec $f(i) \in S_i$ pour tout i (choisir successivement une paire de skis différente pour chaque skieur, la marge de manoeuvre laissée d'au moins $n-1$ autres paires de skis pour chaque skieur garantit que c'est possible). On peut donc appliquer (A1) à n et à cet ensemble S dont le cardinal est au plus $n(2n-1)$. Le temps pris pour déterminer le sous-ensemble S est $O(n((\log m)+n))$: le premier n correspond à la recherche des S_i successifs, $\log m$ est le temps pour déterminer la meilleure paire de skis (par dichotomie dans la liste triée des s_i), et le dernier n correspond à la taille de la fenêtre autour de la meilleure hauteur.
Il reste à voir qu'une solution optimale pour ce sous-ensemble de paires de skis est optimale pour l'ensemble de toutes les paires de skis. On remarque que pour tout $j \notin S$, on a par construction que pour tout i il existe au moins n éléments $s_{j_1}, \ldots, s_{j_n}$ de S_i tels que $|s_j - h_i| \geq |s_{j_k} - h_i|$ $(1 \leq k \leq n)$. (Noter ici l'importance d'avoir pris des paires *de part et d'autre* de la meilleure paire de skis.) Si donc il existait une fonction d'attribution optimale qui atteint j (ainsi possiblement que d'autres valeurs hors de S), on pourrait remplacer sans conflit (cf. ci-dessus) ces valeurs par des valeurs de S, et donc obtenir une autre fonction optimale à valeurs dans S.
La complexité relative à n et S s'exprime comme suit (en y incorporant le temps de recherche des éléments de S) :

$$O(n^2 \log n) + O(n \log n) + O((n^2 - n)n) + O(n((\log m) + n)) = O(n^3 + n \log m) .$$

Ce temps est meilleur que le temps obtenu en (A2) si $n^2 = o(m)$, car alors $O(n^3 + n \log m) = o((m \log m) + (n \log n) + (m-n)n)$. En effet, $(n \log m)/(m \log m)$ et $n^3/(m-n)n$ tendent vers 0 (noter qu'on a fortiori $n = o(m)$). CQFD

4 Partages équilibrés de périmètres

Source. “Algorithmique : exercices corrigés”, Ellipses (1993) (pour le cas $n = 2$).

Soit un polygone dont la suite des longueurs des côtés est $\{a_0, a_1, \ldots, a_n\}$.

(A) Détermlner en un temps linéaire deux indices i, j qui minimisent la valeur absolue de la différence entre les deux portions de périmètre qu'ils déterminent, i.e., qui minimisent (sommations modulo n)

$$|(\sum_i^{j-1} a_l) - (\sum_j^{i-1} a_l)| \, .$$

Preuve. Pour une mise en jambes, l'algorithme le plus naïf est cubique. On fait le calcul de cette valeur absolue $f(i, j)$ pour chaque paire i, j (un tel calcul prend un temps $0(n)$), et on rejette ces calculs au fur et à mesure que l'on trouve une valeur plus petite. Comme il y a $O(n^2)$ paires (i, j), l'algorithme est bien cubique. Il est clair que les calculs des $f(i, j)$ ne sont pas indépendants, on peut donc s'appuyer, pour i fixé, sur $f(i, j)$ pour calculer $f(i, j + 1)$. D'où un algorithme quadratique basé sur les calculs des $\phi(i) = \min_j f(i, j)$. On calcule d'abord le périmètre p (temps $O(n)$). On réécrit des paires (j, S), en partant de $(i, 0)$ en utilisant les règles suivantes :

$$\text{(R1)} \quad \frac{S < p/2}{(j, S) \ \rightarrow \ (j + 1, S + a_j)}$$

$$\text{(R2)} \quad \frac{S \geq p/2}{(j, S) \ \rightarrow !\min(2S - p, p - 2(S - a_{j-1}))}$$

Montrons que cet algorithme est correct, i.e., que $(i, 0) \rightarrow^* !\phi(i)$ pour tout i. On remarque que S croît tant que l'on utilise la règle (R1) et donc que la différence $(p - S) - S = p - 2S$ décroît. Il est inutile de continuer lorsque S a atteint la moitié du périmètre, car la valeur absolue est alors de la forme $2S - p$ et se mettrait à croître. Le choix se cristallise donc autour de $j - 1$ ou j, d'où le min. L'algorithme ci-dessus coûte $O(n)$ (chaque étape de réécriture a un coût constant). Le coût total est linéaire (périmètre) + quadratique (calcul des $\phi(i)$), i.e. quadratique.

Mais les calculs des $\phi(i)$ ne sont pas indépendants. Soit $(i,0) \to^\star (j,S)$, la règle (R3) s'appliquant à (j,S). Si on se déplace de i à $i+1$, on peut directement partir de (j,S) pour le calcul de $\phi(i+1)$! en effet, puisque $S - a_{j-1} < p/2$, on a a fortiori $S - a_i - a_{j-1} < p/2$. Ceci conduit à l'algorithme suivant :

$$\frac{S < p/2}{(i,j,S,A) \ \to \ (i, j+1, S+a_j, A)}$$

$$\frac{S \geq p/2 \quad i < n-1}{(i,j,S,A) \ \to \ (i+1, j, S-a_i, \min(A, 2S-p, p-2(S-a_{j-1})))}$$

$$\frac{S \geq p/2 \quad i = n-1}{(i,j,S,A) \ \to ! \min(A, 2S-p, p-2(S-a_{j-1}))}$$

On part de $(0,0,0,\infty)$. Le calcul s'achève sur un !. Il reste à vérifier que le nombre de pas de réduction est linéaire. Soit

$$(0,0,0,\infty) \to^\star (1, j_1, \_, \_) \to^\star (2, j_2, \_, \_) \to^\star \cdots$$

(où les _ désignent des composantes dont la valeur ne nous intéresse pas ici). Le coût est de $j_1 + 1$ étapes pour aller de $(0,0,0,\infty)$ à $(1, j_1, \_, \_)$, puis de $j_2 - j_1 + 1$ étapes pour aller de $(1, j_1, \_, \_)$ à $(2, j_2, \_, \_)$,..., d'où une sommation $(j_1 + 1) + (j_2 - j_1 + 1) + (j_3 - j_2 + 1) + \cdots$, ce qui donne $O(n)$ (calcul du périmètre) plus un nombre $\leq 2n$ (somme exacte, i.e., non modulo n, des $j_{i+1} - j_i$ – on observe que i fait un tour, et que j est à moins d'un tour de i) plus $O(n)$ (somme des 1).

CQFD

(B) Déterminer en un temps linéaire trois indices i, j, k qui minimisent la différence entre le plus grand "tiers" et le plus petit "tiers" qu'ils déterminent, i.e., qui minimisent (sommations modulo n)

$$\max(\sum_i^{j-1} a_l, \sum_j^{k-1} a_l, \sum_k^{i-1} a_l) - \min(\sum_i^{j-1} a_l, \sum_j^{k-1} a_l, \sum_k^{i-1} a_l) \ .$$

Preuve. Comme plus haut on désigne le périmètre (que l'on calcule en $O(n)$) par p, On note d'abord que, pour tout choix de i, j, k, la plus grande des trois

sommes, que l'on notera respectivement S_1, S_2 et S_3, est $\geq p/3$ (sinon la somme des trois serait $< p$), et que si l'une des sommes est $\geq p/3$, ce n'est pas la plus petite (ceci couvrant le cas dégénéré où toutes les sommes sont égales à $p/3$). De plus, la symétrie circulaire du problème fait que ce n'est pas une restriction que de supposer que la première somme S_1 est la plus grande.
On recherche d'abord une solution telle que S_2 soit la plus petite des trois sommes. Si S_3 est la plus petite, on peut la considérer comme la seconde somme si l'on change d'orientation $(a_{n-1}, a_{n-2}, \ldots, a_0)$. Il suffit donc de deux algorithmes indépendants et identiques à l'ordre de parcours près, et de prendre le meilleur des résultats.
On va montrer qu'une solution optimale (i, j, k) telle que $S_1 \geq S_3 \geq S_2$ vérifie les deux propriétés suivantes :

(P1) $S_1 \geq p/3$ et $S_1 - a_{j-1} < p/3$.

(P2) $S_2 \leq (p - S_1)/2$ et $S_2 + a_k > (p - S_1)/2$.

(P1) On suppose que $S_1 - a_{j-1} \geq p/3$ (noter que ceci implique $S_1 > p/3$ et donc $S_2 < p/3$), et on montre que la coupe $(i, j-1, k)$, de sommes $S'_1 = S_1 - a_{j-1}$, $S'_2 = S_2 + a_{j-1}$ et $S'_3 = S_3$ est meilleure, i.e., que $\max(S'_1, S'_2, S'_3) - \min(S'_1, S'_2, S'_3) < S_1 - S_2$. On remarque d'abord que S'_1 n'est pas la plus petite somme puisque $S'_1 \geq p/3$ (cf. plus haut). Il faut distinguer plusieurs cas (on pose $\Delta' = \max(S'_1, S'_2, S'_3) - \min(S'_1, S'_2, S'_3)$).

$$\begin{array}{lllll}\Delta' & = & S'_1 - S'_2 & = & (S_1 - a_{j-1}) - (S_2 + a_{j-1}) \\ & & & = & (S_1 - S_2) - 2a_{j-1} \ < \ S_1 - S_2\end{array}$$

$$\begin{array}{lllll}\Delta' & = & S'_3 - S'_2 & = & S_3 - (S_2 + a_{j-1}) \\ & & & \leq & S_1 - (S_2 + a_{j-1}) \ = \ (S_1 - S_2) - a_{j-1} \ < \ S_1 - S_2\end{array}$$

$$\begin{array}{lllll}\Delta' & = & S'_1 - S'_3 & = & (S_1 - a_{j-1}) - S_3 \\ & & & \leq & (S_1 - a_{j-1}) - S_2 \ = \ (S_1 - S_2) - a_{j-1} \ < \ S_1 - S_2\end{array}$$

$$\begin{array}{lllll}\Delta' & = & S'_2 - S'_3 & = & (S_2 + a_{j-1}) - S_3 \\ & & & \leq & (S_2 + a_{j-1}) - S_2 \ = \ a_{j-1} \\ & & & < & (p/3 - S_2) + a_{j-1} \ \leq \ S_1 - S_2.\end{array}$$

(P2) Supposons que $S_2 + a_k \leq (p - S_1)/2$. Alors la coupe $(i, j, k+1)$ de sommes $S'_1 = S_1$, $S'_2 = S_2 + a_k$, $S'_3 = S_3 - a_k$ est meilleure. En effet, puisque $S'_2 \leq (p - S_1)/2$, on a $S'_3 \geq (p - S_1)/2$, donc $S'_2 \leq S'_3 < S_3 \leq S_1 = S'_1$, et $\max(S'_1, S'_2, S'_3) - \min(S'_1, S'_2, S'_3) = S'_1 - S'_2 = S_1 - S'_2 < S_1 - S_2$.

On peut maintenant spécifier la moitié de l'algorithme qui recherche une solution du type “grand, puis petit, puis moyen” dans le sens des aiguilles d'une montre. Les propriétés (P1) et (P2) indiquent comment calculer une adaptation du $\phi(i)$ de la question (A) : avancer jusqu'au premier j tel que $\sum_i^{j-1} a_l$ dépasse $p/3$, puis jusqu'au dernier k tel que $\sum_j^{k-1} a_l$ ne dépasse pas la moitié de ce qui reste, et poser $\phi(i) = \sum_i^{j-1} a_l - \sum_j^{k-1} a_l$ si $\sum_k^{i-1} a_l \leq \sum_i^{j-1} a_l$ et $\phi(i) = \infty$ sinon (en effet, dans ce cas, le découpage n'a pas la forme recherchée).
A nouveau, les $\phi(i)$ ne sont pas indépendants. Pour pouvoir adapter l'algorithme (A), il faut montrer que si (i, j, k) et $(i+1, j', k')$ sont des découpages déterminés selon (P1) et (P2) en partant de a_i et a_{i+1}, respectivement, alors $j \leq j'$ et $k \leq k'$. La première inégalité est évidente. Voici pour la seconde. On pose $S_1 = \sum_i^{j-1} a_l$, $S_2 = \sum_j^{k-1} a_l$ et $S'_1 = \sum_{i+1}^{j'-1} a_l$. Il faut vérifier l'inégalité suivante : $S_1 + S_2 - a_i \leq S'_1 + (p - S'_1)/2$, et comme on a $S_2 \leq (p - S_1)/2$ par construction, il suffit de vérifier $S_1 + (p - S_1)/2 - a_i \leq S'_1 + (p - S'_1)/2$, c'est-à-dire $S_1 - 2a_i \leq S'_1$, Or, puisque $j \leq j'$, on a $S_1 - a_i \leq S'_1$ et donc a fortiori $S_1 - 2a_i \leq S'_1$. Tous les feux sont donc verts pour l'algorithme suivant (les états sont de la forme (i, j, k, S, T, A)) :

$$\frac{S < p/3}{(0, j, j, S, 0, \infty) \ \rightarrow\ (0, j+1, j+1, S + a_j, 0, \infty)}$$

$$\frac{S < p/3 \quad i > 0}{(i, j, k, S, T, A) \ \rightarrow\ (i, j+1, k, S + a_j, T - a_j, A)}$$

$$\frac{S \geq p/3 \quad T + a_k \leq (p - S)/2}{(i, j, k, S, T, A) \ \rightarrow\ (i, j, k+1, S, T + a_k, A)}$$

$$\frac{S \geq p/3 \quad T + a_k > (p - S)/2 \quad S < p - S - T \quad i < n-1}{(i, j, k, S, T, A) \ \rightarrow\ (i+1, j, k, S - a_i, T, A)}$$

$$\frac{S \geq p/3 \quad T + a_k > (p - S)/2 \quad S \geq p - S - T \quad i < n-1}{(i, j, k, S, T, A) \ \rightarrow\ (i+1, j, k, S - a_i, T, \min(A, S - T))}$$

$$\frac{S \geq p/3 \quad T + a_k > (p-S)/2 \quad S < p - S - T \quad i = n-1}{(i,j,k,S,T,A) \;\rightarrow !A}$$

$$\frac{S \geq p/3 \quad T + a_k > (p-S)/2 \quad S \geq p - S - T \quad i = n-1}{(i,j,k,S,T,A) \;\rightarrow ! \min(A, S-T)}$$

Les composantes S et T servent au calcul des sommes S_1 et S_2, et la dernière composante A mémorise le $(\max - \min)$ courant. La linéarité de l'algorithme se vérifie comme en (A). Le résultat final est le min des $!x$ fournis par les deux moitiés de l'algorithme. CQFD

5 Calcul de x^n

Source. “Algorithmique : exercices corrigés”, Ellipses (1993) (pour le début) et D. Knuth, “The Art of Computer Programming”, vol. 2, Addison Wesley (1981), section 4.6.3, pour la partie (D).

On suppose que l'on a un ensemble muni d'une loi de composition associative et commutative notée multiplicativement, et on veut calculer x^n à partir de x en faisant des multiplications. Formellement (Γ est une liste de puissances de x) :

$$\frac{}{\emptyset \;\rightarrow\; x} \qquad (\mu)\;\frac{x^i \in \Gamma \quad x^j \in \Gamma}{\Gamma \;\rightarrow\; \Gamma, x^{i+j}}$$

que l'on peut écrire sous forme allégée (mais qui ne tient pas compte du partage, i.e. qui ne rend pas compte de la mise en mémoire des puissances déjà calculées) :

$$\frac{}{x} \qquad (\mu)\;\frac{x^i \quad x^j}{x^{i+j}}$$

L'on souhaite minimiser la taille d'un arbre construit avec ces règles et de conclusion x^n, i.e. minimiser la longueur d'une dérivation $\emptyset \rightarrow^\star \Gamma$ (ou, ce qui revient au même, la longueur de Γ) telle que $x^n \in \Gamma$.

(A1) Donner un algorithme linéaire.

(A2) Donner un algorithme dont le temps est compris entre $\lfloor \log_2 n \rfloor$ et $2\lfloor \log_2 n \rfloor$.

(A3) Discuter ; cet algorithme est-il optimal ?

PREUVE. (A1) L'algorithme consiste à utiliser seulement l'instance suivante de la règle (μ) :

$$\frac{x^i \quad x}{x^{i+1}}$$

(A2) Pour avoir quelque chose de plus efficace, on a recours à l'idée de dichotomie. C'est la méthode dite chinoise, qui s'exprime formellement comme suit :

$$(ch_1)\;\frac{x^i}{x^{2i}} \qquad (ch_2)\;\frac{x^{2i}}{x^{2i+1}}$$

(Noter que dans ce cas et dans le cas précédent, nous avons transformé un système formel non-déterministe en un algorithme : il n'y a qu'une instance de règle dont la racine de l'arbre est conclusion, et ainsi de suite.) La complexité $f(n)$ (nombre de règles utilisées) de cet algorithme est plus facile à voir en regardant la représentation de n en base 2. Tout nombre n de longueur $k+1$ commençant par 1 est compris entre 2^k et 2^{k+1}. On a donc $k = \lfloor \log_2 n \rfloor$. Le nombre j des chiffres égaux à 1, autres que le premier, est égal au nombre d'étapes (ch_2), tandis que k est le nombre d'étapes (ch_1). On peut en effet considérer la représentation binaire de n comme une transcription d'une preuve de x^n selon la méthode chinoise : une application de (ch_1) fait ajouter un chiffre 0 à droite, une application de (ch_2) change ce dernier chiffre 0 en 1. Par exemple, 10110 transcrit la suite $x, x^2, x^4, x^5, x^10, x^{11}, x^{22}$ (procédant de droite à gauche). On a donc

$$\lfloor \log_2 n \rfloor = k \leq f(n) = j + k \leq 2k = 2\lfloor \log_2 n \rfloor \ .$$

La borne inférieure (supérieure) est atteinte pour $n = 2^k$ $(n = 2^{k+1} - 1)$.

(A3) Pour améliorer, on regarde les $2^{k+1} - 1$ (i.e., les cas les pires de (A2)). Pour $k = 3$, on peut trouver mieux (à droite ci-dessous) que l'algorithme chinois (à gauche ci-dessous) :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x}{x^2}}{x^3}}{x^6}}{x^7}}{x^{14}}}{x^{15}} \qquad\qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{x}{x^2} \quad x}{x^3} \quad x^3}{x^6} \quad x^6}{x^{12}} \quad x^3}{x^{15}}$$

ou, respectivement :

$$\begin{aligned} \emptyset \ &\to x \to x, x^2 \to x, x^2, x^3 \to x, x^2, x^3, x^6 \\ &\to x, x^2, x^3, x^6, x^7 \to x, x^2, x^3, x^6, x^7, x^{14} \to x, x^2, x^3, x^6, x^7, x^{14}, x^{15} \end{aligned}$$

$$\begin{aligned} \emptyset \ &\to x \to x, x^2 \to x, x^2, x^3 \to x, x^2, x^3, x^6 \\ &\to x, x^2, x^3, x^6, x^{12} \to x, x^2, x^3, x^6, x^7, x^{12}, x^{15} \ . \end{aligned}$$

CQFD

On va donner une méthode qui s'approche asymptotiquement de l'optimal.

(B1) Montrer que $\lceil \log_2 n \rceil$ est une borne inférieure du coût de n'importe quelle méthode de calcul de x^n.

(B2) Concevoir une méthode de calcul de x^n basée sur la représentation de n en base $m = 2^k$. Choisir k fonction de n de sorte à faire tendre le coût asymptotiquement vers $\log_2 n$.

(B3) Donner des cas particuliers (et les méthodes pour ces cas) où la borne (B1) est atteinte.

Preuve. (B1) On pose $\max(\Gamma) = \max\{i \mid x^i \in \Gamma\}$. On montre l'invariant suivant : si Γ est de longueur $n+1$, et si $x^i \in \Gamma$, alors $i \leq 2^n$. Pour $n = 0$, on a $\Gamma = x$, donc $i = 1 \leq 2^0$. Pour le pas de récurrence, soit $\Gamma \to \Gamma, x^{i+j}$. On a $x^i \leq 2^n$ et $x^j \leq 2^n$ par récurrence. Donc $x^{i+j} \leq 2^n \times 2^n = 2^{n+1}$. Il en résulte que toute méthode α, calculant $x^{\alpha(n)}$ à l'étape n (sans compter $\emptyset \to x$) est telle que $\alpha(n) \leq 2^n$, i.e., $n \geq \log_2 \alpha(n)$.

(B2) Soit $n = d_0 m^t + d_1 m^{t-1} + \cdots + d_t$ ($d_0 > 0$). On obtient d'abord $\emptyset \to^\star \Gamma_0$ avec $x^i \in \Gamma$ pour tout $i < m$ (avec la borne grossière (A1), ceci se fait en $m-2$ étapes, et cette borne est de fait exacte si tous les d_i sont utiles, i.e., $\neq 0$). Puis on dérive

$$\Gamma \to \Gamma, x^{2d_0} \to \cdots \to \Gamma, x^{2d_0}, \ldots, x^{2^k d_0} \to \Gamma, x^{2d_0}, \ldots, x^{md_0}, x^{d_0 m + d_1} = \Gamma_1 \ ,$$

puis $\Gamma_{s-1} \to^\star \Gamma_s$, où $x^{d_0 m^s + d_1 m^{s-1} + \cdots + d_s} \in \Gamma_s$. L'algorithme s'arrête à Γ_t. Le coût total est $\leq (m-2) + (k+1)t$, décomposé comme suit :

$$\begin{array}{ll} \emptyset \to^\star \Gamma_0 & (\leq m-2) \\ \Gamma_0 \to^\star \Gamma_1 & (= k+1) \\ \vdots & \\ \Gamma_{t-1} \to^\star \Gamma_t & (= k+1) \end{array}$$

Il faut exprimer une borne $l(n,k)$ en fonction de n et de k, puis choisir $k = f(n)$ fonction de n tel que $\lim l(n, f(n))/\log_2 n = 1$. Comme $n \geq m^t$, on a $t \leq \log_m n = \log_2 n/k$. On pose donc $l(n,k) = 2^k + ((k+1)\log_2 n)/k$. On a $l(n,k)/\log_2 n = 2^k/\log_2 n + (1 + (1/k))$. Il suffit donc de trouver $f(n)$ tel que $\lim 2^{f(n)}/\log_2 n = 0$ et $\lim f(n) = \infty$, par exemple $2^{f(n)} = \sqrt{\log_2 n}$, i.e., $f(n) = \log_2(\sqrt{\log_2 n}) = (\log_2(\log_2 n))/2$.

(B3) Le calcul de 2^A se fait en $A = \log_2 2^A$ étapes (méthode chinoise, cf. (A2)). Le calcul de $2^A + 2^B$ ($A > B$) se fait en $A + 1 = \lceil \log_2(2^A + 2^B) \rceil$ étapes (méthode chinoise également). CQFD

On explore maintenant une autre méthode.

(C1) Spécifier une méthode de calcul de x^n basée sur la décomposition de n en facteurs premiers.

(C2) Montrer sur des exemples que la méthode des facteurs peut sur une infinité de cas être meilleure que la méthode chinoise, et inversement (on ne prend pas en considération le coût de la recherche de facteurs premiers).

Preuve. (C1)

$$(n \text{ premier}) \quad \frac{x^{n-1} \in \Gamma}{\Gamma \;\rightarrow\; \Gamma, x^n}$$

$$\frac{\emptyset \rightarrow^\star \Gamma, x^p \quad \emptyset \rightarrow^\star \Gamma', x^q}{\Gamma, x^p \rightarrow^\star \Gamma, (\Gamma', x^q)[x^p/x]) = \Gamma, \Gamma'[x^p/x], x^{pq}}$$

Si l'on veut rendre l'algorithme complètement déterministe, on peut par exemple choisir $p =$ le plus petit facteur premier de $n = pq$.

(C2) La méthode des facteurs est meilleure que la méthode chinoise sur tous les nombres de la forme 15×2^k (cf. (A3)). Pour l'autre sens, l'heuristique est de prendre un nombre pour lequel la méthode chinoise est optimale, par exemple $2^i + 1$, par exemple avec $i = 5$. Les deux méthodes produisent respectivement (chinoise en jaut, facteurs en bas) :

$$x, x^2, x^4, x^8, x^{16}, x^{32}, x^{33}$$

$$\begin{array}{rcl} x, x^2, x^3, x^6, x^{12}, x^{24}, x^{30}, x^{33} & = & x, x^2, \Gamma'[x^3/x] \\ & & \text{où } \Gamma' \;=\; x, \Gamma''[x^2/x], x^{11} \\ & & \quad \text{où } \Gamma'' = x, x^2, x^4, x^5 \;. \end{array}$$

Au total, la méthode chinoise (des facteurs) prend 6 (7) étapes. Comme les deux méthodes coïncident sur les puissances de 2, la méthode chinoise gagne sur tous les nombres de la forme 33×2^k. CQFD

On termine par la construction du “power tree” de Knuth. Cet arbre a pour sommets les entiers naturels strictement positifs (avec pour racine 1), sans répétition. Chaque chemin $1, a_1, \ldots, a_m$ correspond à une dérivation

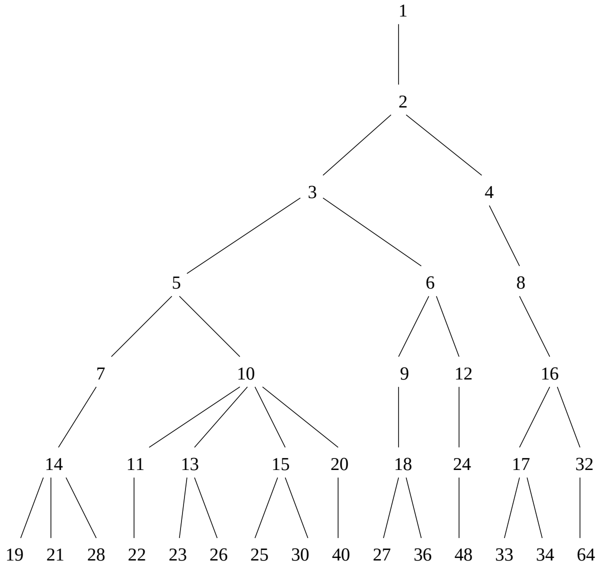
$$\emptyset \;\rightarrow\; x \;\rightarrow\; x, x^{a_1} \;\rightarrow\; \cdots \;\rightarrow\; x, x^{a_1}, \ldots, x^{a_m} \;.$$

On place tous les Γ au fur et à mesure, niveau par niveau et de gauche à droite (i.e., si $\Gamma \to \Gamma, x^n$, alors le sommet correspondant à Γ, x^n est fils du sommet correspondant à Γ), la règle étant de n'insérer ce nouveau sommet que si x^n apparaît pour la première fois dans l'arbre, dans l'ordre de sa construction. On ne prend pas en considération le coût de la construction de l'arbre, ni le coût de la recherche dans l'arbre, mais seulement la longueur du chemin menant à un noeud n, égale à la longueur de la dérivation $\emptyset \to^* \Gamma, x^n$ que l'arbre propose.

(D1) Construire les sept premiers niveaux de l'arbre.

(D2) Discuter ; cet algorithme est-il optimal ?

PREUVE. (D1) Par exemple pour construire les fils du sommet 5 on examine successivement tous les sommets qui le précèdent : 1, 2, 3 et lui-même 5. On a : $5 + 1 = 6$, déjà obtenu ; $5 + 2 = 7$, ajouté à l'arbre ; $5 + 3 = 8$, déjà obtenu ; $5 + 5 = 10$, ajouté à l'arbre :



(D2) L'algorithme n'est pas optimal : ne pas répéter une valeur déjà obtenue n n'est pas génant pour le calcul de x^n mais pourrait être utile pour une valeur ultérieure. Les plus petits exemples sont $n = 77, 154, 233$. Le plus petit entier pour lequel le power tree est meilleur que la chaine chinoise et la méthode des facteurs est $n = 23$ (en six étapes sur la figure contre sept pour les deux autres méthodes), alors que le plus petit entier pour lequel la méthode des facteurs bat le power tree est $n = 19879 = 103 \times 193$. Voir le livre de Knuth pour de plus amples statistiques ... CQFD

6 Coloration

Source. D. West, "Introduction to graph theory", Prentice Hall (1996) pour les parties (A) et (B), et H.S. Wilf, "Algorithmes et complexité", Masson (1989) pour la partie (C).

On s'intéresse à quelques algorithmes simples pour colorier les sommets d'un graphe. Règle du jeu : deux sommets adjacents ne doivent pas recevoir la même couleur.

2-coloriage. Les graphes 2-coloriables (qu'on peut colorier avec 2 couleurs) sont les graphes bipartis : $G = (V, E)$ est biparti s'il existe une partition de V en deux sous-ensembles B (bleu) et R (rouge) tels que toute arête de E relie un sommet bleu et un sommet rouge.

(A1) Proposer un algorithme pour vérifier si un graphe est 2-coloriable.

(A2) Montrer qu'un graphe est 2-coloriable si et seulement si il n'admet pas de cycle de longueur impaire.

PREUVE. (A1) On peut supposer $G = (V, E)$ connexe sans perte de généralité. Une solution en $O(|V| + |E|)$: prendre un sommet au hasard et le marquer bleu ; tant qu'il existe des sommets non marqués, prendre un sommet adjacent à l'un des sommets déjà marqués et le marquer de l'autre couleur, si c'est possible (i.e. s'il n'est pas également adjacent à un autre sommet déjà colorié de l'autre couleur) ; s'arrêter sinon en signalant l'échec.

(A2) Le sens $\Rightarrow$ est évident. Pour la réciproque, on utilise l'algorithme précédent : on ne sera jamais en situation d'échec, car sinon on aurait fait apparaître un cycle de longueur impaire. CQFD

Heuristiques gloutonnes. Le cas des graphes 2-coloriables est réglé. Hélas, le problème de n-coloriabilité est NP-complet pour $n \geq 3$, ce qui motive l'étude de quelques heuristiques de coloriage. Désormais, les couleurs sont représentées par des entiers $1, 2, \ldots, K, \ldots$ Le degré $\delta(v)$ d'un sommet v est le nombre de ses voisins, et le degré $\Delta(G)$ d'un graphe G est le maximum des degrés de ses sommets.

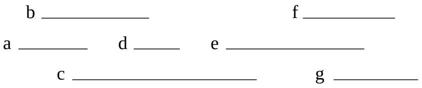
(B1) Etant donné un ordre arbitraire $v_1, v_2, \ldots, v_n$ des sommets de $G = (V, E)$, l'algorithme glouton colorie les sommets dans cet ordre en attribuant à chaque sommet la plus petite couleur disponible (non déjà utilisée pour les voisins). Montrer que le nombre de couleurs utilisées $gl(G)$ vérifie $gl(G) \leq \Delta(G) + 1$.

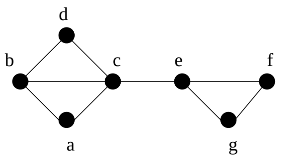
(B2) On pose $d_i = \delta(v_i)$. Montrer qu'on a $gl(G) \leq \max_{i=1}^{n} \min(d_i + 1, i)$. Quelle manière d'énumérer préalablement les sommets du graphe cette borne affinée suggère-t-elle ?

Preuve. (B1) Au moment où on traite un sommet donné, il a au plus $\Delta(G)$ voisins déjà coloriés, donc l'algorithme glouton n'est jamais forcé d'utiliser plus de $\Delta(G) + 1$ couleurs.

(B2) Au moment où on colorie le i-ème sommet v_i, il a au plus $\min(d_i, i-1)$ voisins qui ont déjà été coloriés, et donc sa propre couleur est au plus $1 + \min(d_i, i-1) = \min(d_i + 1, i)$. En prenant le maximum sur i de ces valeurs, on obtient la borne demandée. Cette borne suggère de se débarrasser d'abord des plus gros degrés, qui disparaissent ainsi de la complexité tant que $\min(d_i + 1, i) = i$. Comme on ne sait pas a priori jusqu'à quand privilégier les grands degrés, on trie l'ensemble des sommets par degrés décroissants. CQFD

Nous allons voir que l'algorithme glouton est optimal pour les graphes d'intervalles. Etant donnée une famille d'intervalles (disons sur la droite réelle), on définit un graphe dont les sommets sont les intervalles, et dont les arêtes relient les sommets représentant les intervalles qui s'intersectent. Voici un exemple :





(B3) Montrer que l'algorithme glouton colorie le graphe de manière optimale si l'on énumère les sommets d'un graphe d'intervalles G selon l'ordre donné par les extrémités gauches des intervalles, à savoir a, b, c, d, e, f, g sur l'exemple.

Preuve. Sur l'exemple on obtient le coloriage $1, 2, 3, 1, 1, 2, 3$ qui est bien optimal. Dans le cas général, exécutons l'algorithme glouton avec l'ordre spécifié des sommets. Supposons que le sommet v recoive la couleur maximale k. Pourquoi v ne reçoit-il pas une couleur plus petite ? Parce que l'extrémité de gauche a de son intervalle appartient aussi à des intervalles qui ont déjà reçu les couleurs 1 à $k-1$. Ces intervalles ont donc tous le point a en commun, si bien que G possède une clique (un sous-graphe complet) de taille k. Le cardinal maximal d'une clique de

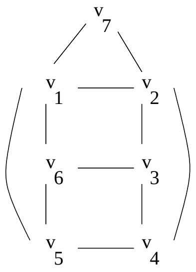
G est donc supérieur ou égal à k. Comme tous les sommets d'une clique doivent recevoir une couleur différente, on voit que l'on ne peut pas faire mieux que k couleurs. L'algorithme glouton est donc optimal. CQFD

Voici une dernière variante intéressante de l'algorithme glouton. On définit le degré-couleur d'un sommet comme le nombre de couleurs déjà utilisées pour colorier ses voisins (attention, ce n'est pas le nombre de voisins déjà coloriés, car certains peuvent avoir reçu la même couleur). Initialement, le degré-couleur de chaque sommet est nul, puis celui-ci évolue au cours de l'algorithme, jusqu'à ce que le sommet soit finalement colorié. Plus précisément, voici l'algorithme de Brelaz :

Initialisation. Trier les sommets par degrés décroissants. Choisir un sommet de degré maximal et lui attribuer la couleur 1.

Régime permanent. Tant qu'il reste des sommets non coloriés, choisir un sommet non-colorié de degré-couleur maximal (en cas d'égalité, prendre parmi ceux-ci un sommet de degré maximal), et le colorier avec la plus petite couleur admissible.

(B4) Faire tourner l'algorithme précédent sur l'exemple ci-dessous :



(B5) Montrer que cet algorithme n'utilise que deux couleurs pour colorier les graphes bipartis.

Preuve. (B4) On prend au départ un sommet de degré maximal, par exemple v_1, et on lui donne la couleur 1. Le degré-couleur de v_2, v_5, v_6 et v_7 passe à 1, on choisit v_2 qui est de degré maximal parmi ces trois sommets, et on lui donne la couleur 2. Maintenant, v_7 est le seul sommet de degré-couleur 2, on le choisit et on lui attribue la couleur 3. Tous les sommets restants (non coloriés) ont le même degré-couleur 1 et le même degré 3, on choisit v_3 au hasard, et on lui donne la couleur 1. Puis v_4, de degré-couleur 2, reçoit la couleur 3. Enfin v_5 reçoit la couleur 2 et v_6 la couleur 3. Le graphe est 3-colorié, c'est optimal.

(B5) On considére un graphe biparti $G = (V, E)$, $V = B \cup R$: toute arête de E relie un sommet de B (bleu) et un sommet de R (rouge). Remarquons d'abord que l'algorithme glouton défini en (B1) ou en (B2) peut se tromper : par exemple soit G avec $B = \{b_1, b_2, b_3\}$, $R = \{r_1, r_2, r_3\}$, et les six arêtes (b_i, r_i), $1 \leq i \leq 3$, (b_1, r_2), (b_2, r_3) et (b_3, r_1). Tous les sommets sont de degré 2. Si on commence par colorier un sommet de B, disons b_1, puis un sommet non adjacent de R, r_3, avec la même couleur 1, on ne pourra pas terminer avec seulement 2 couleurs. Par contre, en utilisant la variante de Brelaz avec les degrés-couleur sur ce petit exemple, après b_1 on doit colorier soit r_1 soit r_2, et on ne se trompe pas. Pour le cas général, considérons $G = (V, E)$, $V = B \cup R$ connexe biparti, et colorions le avec l'algorithme de Brelaz. On commence par colorier un sommet, disons, de B, en bleu. Puis on colorie nécessairement un sommet adjacent (de degré-couleur 1), donc de R, en rouge. Par la suite, on ne colorie que des sommets de degré-couleur égal à 1, c'est l'invariant qui donne le résultat.

CQFD

Sur le nombre de coloriages d'un graphe. L'objet de cette partie est de démontrer le résultat surprenant suivant : le nombre moyen de K-coloriages d'un graphe de n sommets est borné par une constante indépendante de n. Commençons par quelques résultats préliminaires.

(C1) Soient K nombres positifs $s_1, s_2, \ldots, s_K$ dont la somme est L. Montrer que la somme de leurs carrés vaut au moins $\frac{L^2}{K}$.

(C2) Soit C une des K^L façons de colorier avec K couleurs un ensemble de L sommets virtuels. Montrer que le nombre de graphes G dont l'ensemble des sommets est cet ensemble et pour lesquels C est un coloriage correct est au plus $2^{\frac{L^2}{2}(1-\frac{1}{K})}$.

(C3) Le nombre total de coloriages corrects avec K couleurs de tous les graphes de L sommets est au plus $K^L \times 2^{\frac{L^2}{2}(1-\frac{1}{K})}$.

Preuve. (C1) $0 \leq \sum_{i=1}^{K}(s_i - L/K)^2 = \sum_{i=1}^{K} s_i^2 - L^2/K$ (en utilisant $\sum_{i=1}^{K} s_i = L$).

(C2) Dans cette question, au lieu de se demander combien un graphe donné peut avoir de coloriages corrects, on se demande combien de graphes admettent un coloriage donné C. Supposons que dans C, s_1 sommets aient la couleur 1, s_2 sommets la couleur 2, ..., s_K sommets la couleur K, avec $s_1 + s_2 + \cdots + s_K = L$. Si un graphe G admet C comme coloriage, ses arêtes ne peuvent être qu'entre sommets de couleurs différentes. Le nombre des arêtes de G est donc majoré par

$$s_1 s_2 + s_1 s_3 + \cdots + s_1 s_K + s_2 s_3 + \cdots + s_2 s_K + \cdots + s_{K-1} s_K$$

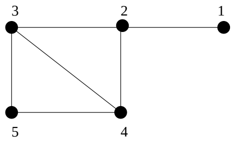
que nous pouvons estimer par

$$\begin{aligned}\textstyle\sum_{1\le i<j\le K} s_i s_j \;=\; \frac{1}{2}\sum_{i\neq j} s_i s_j \;&=\; \frac{1}{2}(\sum_{i=1}^{K} s_i)^2 - \frac{1}{2}\sum_{i=1}^{K} s_i^2\\ &\le\; \frac{L^2}{2} - \frac{L^2}{2K} \;=\; E \qquad \text{(par (C1))}\ .\end{aligned}$$

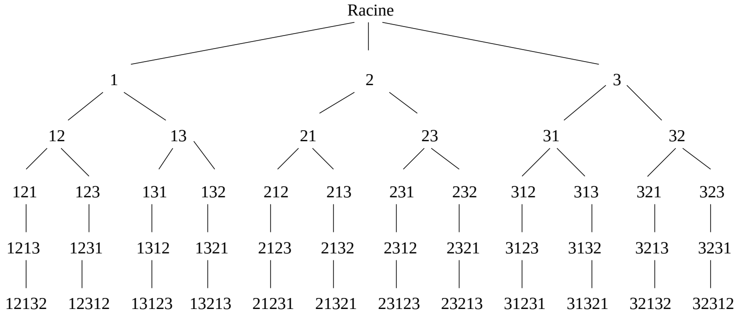
Chaque arête possible pouvant être présente dans G ou non, le nombre de graphes candidats est au plus 2^E, la quantité cherchée.

(C3) On compte les paires (G, C) pour lesquelles le graphe G a L sommets et C est un coloriage correct de G. Si on fixe C, on a 2^E graphes au plus. Il y a K^L coloriages possibles (K choix pour chacun des L sommets), d'où le résultat. CQFD

On introduit maintenant la recherche "backtracking" pour le coloriage des graphes. On construit l'arbre de toutes les possibilités de coloriage, où l'étiquette de chaque noeud décrit les couleurs déjà affectées aux premiers sommets. Voici un exemple de graphe :



et son arbre "backtracking"de coloriage avec $K = 3$ couleurs :



Dans l'arbre, l'étiquette 212 signifie que les trois sommets 1, 2 et 3 ont été respectivement coloriés en 2, 1 et 2. Le nom de "backtracking" vient de l'algorithme de parcours : on descend tant qu'on peut dans l'arbre, et après la fin d'une exploration (couronnée de succès ou terminant par un échec) on remonte l'arbre le moins possible pour poursuivre l'exploration.

Le niveau L de l'arbre de coloriage avec K couleurs d'un graphe G (la racine est au niveau 0) représente tous les coloriages corrects en K couleurs du sous-graphe $H_L(G)$ induit par les L premiers sommets de G. Le nombre de noeuds à ce niveau est donc égal au nombre de coloriages corrects de $H_L(G)$ avec K couleurs : on note ce nombre $P(K, H_L(G))$ (P représente le polynôme chromatique, mais c'est une autre histoire).

(C4) Montrer que le nombre moyen $A(n, K)$ de noeuds dans un arbre de coloriage "backtracking" pour colorier les graphes de n sommets avec K couleurs est borné par une constante $h(K)$ indépendante de n.

PREUVE. On en déduira en particulier que le nombre moyen de K-coloriages d'un graphe de n sommets (qui correspondent aux noeuds du dernier niveau de l'arbre) est lui-même borné par $h(K)$.
On note $\mathcal{G}_n$ l'ensemble des graphes à n sommets, dont le cardinal est $2^{C_n^2}$, puisqu'il y a C_n^2 arêtes possibles. On a successivement (# veut dire "cardinal de" ou "nombre de") :

$$\begin{array}{rcl} A(n,K) & = & \frac{1}{\#\mathcal{G}_n} \sum_{G\in\mathcal{G}_n} (\#\text{ noeuds dans l'arbre de } G) \\ & = & 2^{-C_n^2} \sum_{G\in\mathcal{G}_n} (\sum_{L=0}^{n} (\#\text{ noeuds au niveau } L)) \\ & = & 2^{-C_n^2} \sum_{G\in\mathcal{G}_n} (\sum_{L=0}^{n} P(K, H_L(G))) \\ & = & 2^{-C_n^2} \sum_{L=0}^{n} (\sum_{G\in\mathcal{G}_n} P(K, H_L(G))) \ . \end{array}$$

Fixons une valeur de L et considérons la somme interne, qui porte sur tous les graphes de n sommets, On remarque que beaucoup de graphes G de n sommets ont le même $H_L(G)$; exactement $2^{C_n^2 - C_L^2}$. On a donc

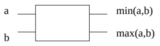
$$\begin{array}{rcll} A(n,K) & = & 2^{-C_n^2} \sum_{L=0}^{n} 2^{C_n^2 - C_L^2} \left\{ \sum_{H\in\mathcal{G}_L} P(K,H) \right\} & \\ & = & \sum_{L=0}^{n} 2^{-C_L^2} \left\{ \sum_{H\in\mathcal{G}_L} P(K,H) \right\} & \\ & \leq & \sum_{L=0}^{n} 2^{-C_L^2} K^L 2^{\frac{L^2}{2}(1-\frac{1}{K})} & \text{(par (C3))} \\ & = & \sum_{L=0}^{n} K^L 2^{\frac{L}{2}} 2^{-\frac{L^2}{2K}} & \\ & \leq & \sum_{L=0}^{+\infty} K^L 2^{\frac{L}{2}} 2^{-\frac{L^2}{2K}} & \end{array}$$

La série infiinie converge, donc la quantité $A(n, K)$ est bornée pour tout n par la somme de la série $h(K)$. CQFD

7 Réseaux de tri

Source. S. Akl, “The design and anlysis of parallel algorithms”, Prentice Hall (1989), chapitre 3, et A. Gibbons et W. Rytter, “Efficient parallel algorithms”, Cambridge University Press (1988), chapitre 5.

Le but du jeu est de disposer et d’interconnecter des comparateurs-échangeurs (comparateurs, pour faire bref) qui prennent deux nombres en entrée et renvoient le plus petit sur la première sortie et le plus grand sur la seconde sortie (voir figure ci-dessous) pour fusionner des listes, puis pour trier des listes.



On recherche une architecture qui ne dépend que de la taille des listes, et non des valeurs de leurs éléments. On note $\textsc{tri}(c_1, c_2 \dots, c_n)$ la liste triée des c_i et on écrit $\textsc{triée}(c_1, c_2 \dots, c_n)$ pour $c_1 \leq c_2 \leq \cdots \leq c_n$. Sous les hypothèses $\textsc{triée}(a_1, \dots, a_n)$ et $\textsc{triée}(b_1, \dots, b_n)$, on note

$$\textsc{fusion}((a_1, \dots, a_n), (b_1, \dots, b_n)) = \textsc{tri}(a_1, \dots, a_n, b_1, \dots, b_n) \ .$$

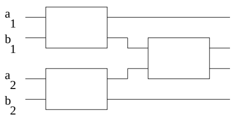
(A1) Concevoir un réseau $\underline{\textsc{fusion}_1}$, ($\underline{\textsc{fusion}_2}$) à partir de comparateurs pour la fusion de deux listes triées de longueur 2 (2^2).

(A2) Montrer la propriété suivante, pour toute paire de listes

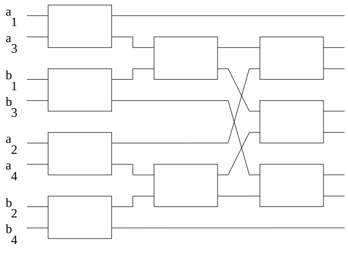
$$\frac{\begin{array}{c}\textsc{triée}(a_1, \dots, a_{2n}) \quad \textsc{triée}(b_1, \dots, b_{2n}) \\ \textsc{fusion}((a_1, a_3, \dots, a_{2n-1}), (b_1, b_3, \dots, b_{2n-1})) = (d_1, \dots, d_{2n}) \\ \textsc{fusion}((a_2, a_4, \dots, a_{2n}), (b_2, b_4, \dots, b_{2n})) = (e_1, \dots, e_{2n})\end{array}}{\textsc{triée}(d_1, \min(d_2, e_1), \max(d_2, e_1), \dots, \min(d_{2n}, e_{2n-1}), \max(d_{2n}, e_{2n-1}), e_{2n})}$$

(A3) Définir par récurrence un réseau $\underline{\textsc{fusion}_n}$ pour la fusion de deux listes de longueur 2^n. Combien de comparateurs comprend ce réseau, et quel en est le temps de calcul ? Combien de comparaisons sont effectuées ?

Preuve. (A1) Voici un réseau $\underline{\textsc{fusion}_1}$ pour fusionner deux listes triées de taille 2 :



Le lecteur se convaincra facilement de la validité de ce réseau. Pour le réseau FUSION$_2$, on utilise deux copies du réseau FUSION$_1$ précédent, et on ajoute une rangée de 3 comparateurs :



(Ces architectures sont des cas particuliers de de la construction (A3) justifiée par (A2).)

(A2) Dans la liste conclusion, d_1 apparaît en première position, et c'est bien le plus petit élément de la liste finale ; de même, e_{2n} est à sa place en dernière position. Pour le cas général, nous allons montrer que d_i et e_{i-1} apparaissent en position $2i-2$ ou $2i-1$ dans la liste conclusion. Par construction, d_i domine les $i-1$ éléments $d_1, \ldots, d_{i-1}$. On a $d_1 = a_1$ ou $d_1 = b_1$, disons $d_1 = a_1$. On distingue deux cas. Si b_1 ne figure pas dans $d_1, \ldots, d_i$, alors les $i-1$ éléments d_j $(1 < j \leq i)$ sont de la forme a_{2k+1}, avec $k \geq 1$. Comme $a_{2k} \leq a_{2k+1}$, on en déduit que d_i domine $2(i-1)$ éléments de la liste $a_1, \ldots, a_{2n}, b_1, \ldots, b_{2n}$. Si b_1 apparaît dans $d_1, \ldots, d_i$, alors la borne est $(i-1)+(i-2)$ (b_1 n'a pas de prédécesseur). On montre de même que d_i est dominé par $4n-(2i-3)-2$ ou $4n-(2i-3)-1$ éléments. Et de même pour e_{i-1} (dans les deux sens). Ceci montre bien que la position de d_i et e_{i-1} dans la liste triée est bien déterminée à une transposition près.

(A3) Pour constituer FUSION$_n$, on dispose deux réseaux FUSION$_{n-1}$ et on connecte leurs sorties à l'aide de $2n-1$ comparateurs, comme suggéré par (A2). C'est ce qui explique a posteriori la rangée de trois comparateurs ajoutée aux deux réseaux FUSION$_1$ pour construire le réseau FUSION$_2$ en (A1).
Le temps de calcul t_n est défini comme le nombre maximal de comparateurs que doit traverser une donnée entre l'entrée et la sortie du réseau. Ainsi, $t_2 = 2$ car certaines données traversent deux comparateurs (bien que d'autres n'en traversent qu'un seul), et $t_3 = 3$. Bien sûr, plusieurs comparateurs peuvent être actifs au même moment (c'est l'idée même du parallélisme !).
Le temps de calcul t_n et le nombre p_n comparateurs sont donnés par les récurrences

suivantes :

$$\begin{array}{lll} t_1 = 2 & t_n = t_{n-1} + 1 & (\text{i.e.}, t_n = n + 1) \\ p_1 = 3 & p_n = 2p_{n-1} + 2^n - 1 & (\text{i.e.}, p_n = 2^n n + 1) \end{array}$$

Ces mesures, exprimées en la taille de chacune des deux listes d'entrée du réseau, sont en $\log n$ et en $n(\log n)$, respectivement. Le rendement du réseau est faible : si on multiplie le nombre de comparateurs par le temps de calcul, on a $p_n \times t_n$ de l'ordre de $n(\log n)^2$, ce qui est moins bien que pour une fusion séquentielle linéaire : un seul comparateur et un temps n. Mais le temps d'exécution est très rapide.
Une parenthèse : le faible rendement s'explique par le fait que chaque comparateur ne fonctionne qu'une seule fois au cours de la fusion. On pourrait augmenter le rendement du réseau en lui demandant de fusionner plusieurs paires de listes différentes. Le même réseau peut en effet commencer à traiter deux nouvelles listes à chaque unité de temps, on dit alors qu'il fonctionne en mode pipeline. Après un délai d'initialisation égal à la profondeur du réseau (qui est un autre nom pour t_n), tous les comparateurs sont actifs à chaque top, et deux nouvelles listes fusionnées sont délivrées en sortie à chaque top également (on dit que la période du réseau est 1). CQFD

(B) Construire par récurrence un réseau $\underline{\textsc{tri}_n}$ pour trier 2^n éléments. Analyser la complexité (temps de calcul, nombre de processeurs, coût).

Preuve. Il suffit de placer deux réseaux $\underline{\textsc{tri}_{n-1}}$ devant un réseau $\underline{\textsc{fusion}_{n-1}}$. Le temps de calcul t'_n et le nombre p'_n de processeurs sont donnés par

$$\begin{array}{lll} t'_1 = 1 & t_n = t'_{n-1} + t_{n-1} = t'_{n-1} + n & (\text{donc } t'_n = O(n^2)) \\ p'_1 = 1 & p'_n = 2p'_{n-1} + 2^{n-1}(n-1) + 1 & (\text{donc } p'_n = O(2^n n^2) + O(2^n) = O(2^n n^2)) \end{array}$$

(pour la seconde récurrence, on a $p'_n = 2^{n-1}(1 + 2 + 3 + \cdots + (n-1)) + (1 + 2 + 4 + \cdots + 2^{n-1})$). Donc, en la taille de la liste à trier, le temps est en $\log^2 n$, le nombre de processeur est en $n(\log^2 n)$, et le coût est en $n(\log^4 n)$. La conclusion est que la taille de l'architecture est prohibitive par rapport au gain en temps.
CQFD

On donne maintenant une technique de preuve différente pour la preuve de correction des réseaux construits en (A3). On dit qu'une suite est de type 0-1 si elle n'est constituée que de 0 et de 1.

(C1) Montrer qu'un réseau de comparateurs implante correctement le tri si et seulement si il calcule bien cette fonction pour toutes suites de type 0-1 en entrée.

(C2) Utiliser (C1) pour prouver la correction des réseaux $\underline{fusion_n}$.

PREUVE. (C1) Soit f une fonction (non strictement) croissante. Un comparateur se comporte de la même manière sur (x_1, x_2) et sur $(f(x_1), f(x_2))$. On considère maintenant un réseau R fixé, appliqué à une suite donnée $(x_1, \ldots, x_n)$. La position finale de l'entrée x_i, i.e., le fil de sortie du réseau où cet x_i aboutit, ne dépend pas de la valeur de x_i mais de sa position relative par rapport aux autres x_j. Donc, si le réseau est appliqué à $(f(x_1), \ldots, f(x_n))$, $f(x_i)$ sort par le même fil de sortie que x_i plus haut. Supposons maintenant que R ne calcule pas correctement la fonction de tri. Il existe donc une suite $(x_1, \ldots, x_n)$, et il existe i_1 et i_2 tels que x_{i_1} et x_{i_2} sortent de R sur deux sorties consécutives, mais $x_{i_1} > x_{i_2}$, Il suffit de définir une fonction monotone $f : \{x_1, \ldots, x_n\} \rightarrow \{0, 1\}$ telle que $f(x_{i_1}) = 1$ et $f(x_{i_2}) = 0$, par exemple :

$$f(x) = \begin{cases} 0 & \text{si } x \leq x_{i_2} \\ 1 & \text{si } x > x_{i_2} \end{cases}$$

Alors R ne calcule pas correctement le tri de $(f(x_1), \ldots, f(x_n))$, puisque on trouve 1 et 0 dans cet ordre sur deux sorties consécutives. Or $(f(x_1), \ldots, f(x_n))$ est une suite de type 0-1, d'oû la conclusion.

(C2) On reprend la preuve de (A3), dans un cadre simplifié. Toute suite triée $(x_1, \ldots, x_m)$ de type 0-1 est de la forme $O^r 1^{m-r}$ (notation $r = \text{ZÉROS}(x_1, \ldots, x_m)$). On pose $p = \text{ZÉROS}(a_1, \ldots, a_{2n})$ et $q = \text{ZÉROS}(b_1, \ldots, b_{2n})$. On distingue plusieurs cas :

- $p = 2p'$ et $q = 2q'$: Dans ce cas, on a

$$\text{ZÉROS}(d_1, \ldots, d_{2n}) = p' + q' = \text{ZÉROS}(e_1, \ldots, e_{2n}) \; .$$

  Les suites $d_1, \ldots, d_{2n}$ et $e_1, \ldots, e_{2n}$ ont le même nombre de 0 (et de 1). Mais il y a un décalage, d_1 (resp : e_{2n}) n'est pas concerné par la rangée des $2n - 1$ comparateurs finaux : le i-ème comparateur compare d_{i+1} avec e_i. La suite des couples d'entrée des comparateurs finaux est formée de $p' + q' - 1$ paires de 0, suivies d'une paire 10 (le premier 1 de $(d_1, \ldots, d_{2n})$ et le dernier 0 de $(e_1, \ldots, e_{2n})$), suivie de paires de 1. On obtient la suite triée grâce au $(p' + q')$-ème comparateur, qui est le seul “utile”.

- $p = 2p'$ et $q = 2q' - 1$. On a maintenant

  $$\textsc{zéros}(d_1, \ldots, d_{2n}) = p' + q' \qquad \textsc{zéros}(e_1, \ldots, e_{2n}) = p' + q' - 1\ .$$

  La suite des couples d'entrée des comparateurs finaux est formée de $p' + q' - 1$ paires de 0, suivies de paires de 1. Aucun des $2n - 1$ comparateurs ne fait un travail utile, et la liste finale $O^{p+q}1^{4n-p-q}$ est obtenue par simple entrelacement.
- $p = 2p' - 1$ et $q = 2q'$. Ce cas est similaire au précédent.
- $p = 2p' - 1$ et $q = 2q' - 1$. Cette fois on a

  $$\textsc{zéros}(d_1, \ldots, d_{2n}) = p' + q' \qquad \textsc{zéros}(e_1, \ldots, e_{2n}) = p' + q' - 2\ .$$

  La suite des couples d'entrée des comparateurs finaux est formée de paires de 0, suivies d'une paire 01, suivie de paires de 1. Aucun des $2n - 1$ comparateurs ne fait un travail utile. CQFD

8 Macro-communications sur un anneau

Source. M. Gengler et al., “Initiation au Parallélisme”, Masson 1996.

Le principe est que tous les processeurs exécutent le même programme (architecture SPMD), et ont leur mémoire locale. Les processeurs sont interconnectés sur un anneau orienté, ils peuvent envoyer au successeur le contenu d'une adresse `x` (instruction `send(x)`), et recevoir du prédécesseur une valeur et la placer à l'adresse `x` (instruction `receive(x)`). Le temps d'un envoi ou d'une réception est $\beta + L\tau$, où β et τ sont deux constantes (start-up et inverse de la bande passante), et où L est la taille du message (ou de la donnée) envoyé. On peut recevoir et envoyer en parallèle des données indépendantes (i.e., deux processeurs différents peuvent envoyer ou recevoir simultanément, et un même processeur peut exécuter `send(x)` et `receive(y)` en même temps pour peu que `x` et `y` soient distincts). Chaque processeur peut connaître son numéro (notation `my_num()`).

- **(A1)** Soit `k` un processeur fixé. Ecrire un programme qui réalise l'envoi par `k` d'un même message de longeur L à tous les autres processeurs (broadcasting).
- **(A2)** Même question, le message étant maintenant découpé en `r` morceaux d'égale longueur $L/$`r`.
- **(A3)** Donner le temps de calcul de ces deux programmes (et asymptotiquement pour le second).

Preuve. (A1) Au début du programme, le message est stocké à l'adresse `x` du processeur `k`. A la fin du programme, le message est stocké à l'adresse `x` de chaque processeur. Il n'y a pas de parallélisme possible : les `send` et les `receive` ne sont pas indépendants.

```
if ny_num()=k
   then send(x)
   else if my_num() = k-1 mod n
      then receive(x)
      else (receive(x) ; send(x))
```

(A2) Au début du programme, les `r` morceaux du message sont stockés aux adresses `x[1]`,…,`x[r]` du processeur `k`. A la fin du programme, le message est stocké dans tous les processeurs aux mêmes adresses. Le principe est celui du pipeline : pendant qu'un processeur reçoit une portion du message, il envoie la portion précédente au processeur suivant, qui lui-même envoie la portion précédente au processeur suivant, etc.….

```
q=my_num() if q = k
   then for i=1 to r send(x[i])
   else if q = (k-1) mod n
      then for i=1 to r receive(x[i])
      else receive(x[1]) ;
         (for i=1 to r-1 (send(x[i]) // receive(x[i+1]))) ;
         send(x[r])
```

(A3) On examine d'abord le premier algorithme (A1). Il y a `n-1` communications successives, d'où un temps $(\mathtt{n}-1)(\beta+L\tau)$. Pour le second algorithme, il suffit de regarder le temps qu'il faut pour que l'ensemble du message soit arrivé au dernier processeur `(k-1 mod n)`. Il faut d'abord le nombre `n-1` d'étapes de communication pour qu'arrive le premier morceau ; puis les autres `r-1` morceaux arrivent immédiatement les uns derrière les autres. Donc en tout un temps de $((\mathtt{n}-2)+\mathtt{r})(\beta+(L/\mathtt{r})\tau)$, que l'on cherche à minimiser. C'est un problème de la forme $(c+ax)(d+b/x)$ avec ad et cb positifs. Ce problème se ramène à minimiser $adx+(cb/x)$, puis à minimiser $(\sqrt{adx}-\sqrt{cb/x})^2$. D'oû $x=\sqrt{cb/ad}$. Et la valeur minimisée est

$$\begin{aligned}((c+ax)(d+b/x))[\sqrt{cb/ad}/x] &= ((c/d)(d+\sqrt{dba/c}))(d+\sqrt{adb/c}) \\ &= (\sqrt{c/d}(d+\sqrt{dba/c}))^2 \\ &= (\sqrt{cd}+\sqrt{ba})^2 \ .\end{aligned}$$

Donc, ici, le `r` optimal est $\sqrt{L(\mathtt{n}-2)\tau/\beta}$ et le temps optimal est

$$(\sqrt{(\mathtt{n}-2)\beta}+\sqrt{L\tau})^2 \ .$$

Dans cette expression, $\mathtt{n}, \beta$ et τ sont fixes. Donc, pour de longs messages, l'expression devient équivalente à $L\tau$: le facteur `n` a disparu. CQFD

On étudie maintenant la diffusion personnalisée (scattering). Le processeur `k` envoie un message différent à chaque autre processeur. Tous ces messages ont la même longueur L.

(B) Ecrire un programme pour la diffusion personnalisée permettant une bonne parallélisation des communications. En évaluer le coût.

PREUVE. Au début du programme, le processeur `k` contient le message pour le processeur `j` à l'adresse `x[j]`. A la fin du programme, chaque processeur a son message à l'adresse `x` (par uniformité, il y a aussi un message `x[k]` pour le processeur `k`). Chaque processeur a aussi une adresse `temp` utilisée pour permettre la parallélisation de l'émission d'un message et de la réception du suivant. L'idée de l'algorithme est d'envoyer les messages en pipeline, en commençant par le message destiné au processeur le plus éloigné : ainsi le temps total est de $(\mathtt{n}-1)(\beta + L\tau)$.

```
q=my_num() if q=k
 then (for i=1 to n-1 send(x[k-i mod n])) ; x:=x[k]
 else
   receive(x) ;
   for i=1 to k-q-1 mod n ((send(x) // receive(temp)) ; x:=temp
```

CQFD

La question suivante est une illustration de la programmation sur un anneau,

(C) Programmer la multiplication d'une matrice de dimension `(m,m)` par un vecteur $(b_1, \ldots, b_{\mathtt{m}})$, de résultat $(c_1, \ldots, c_{\mathtt{m}})$, sur un anneau de `n` processeurs ayant chacun en mémoire `r=m/n` lignes de la matrice.

Preuve. Chaque processeur a en mémoire `r` lignes de la matrice, rangées dans une matrice `a` de dimension `(r,m)` (plus précisément, le processeur `i` contient les lignes `ir+1` à `(i+1)r` de la matrice). Au début du programme, chaque processeur `i` contient les éléments $b_{\mathtt{ir}+1}, \ldots, b_{(\mathtt{i}+1)\mathtt{r}}$ dans un vecteur `x` de dimension `r`.

```
for i=0 to n-1
   ((for p=1 to r
      for q=1 to r
         y[p]:=y[p]+(a[p,(ir+q) mod n]*x[q])) //
   send(x) //
   receive(temp)) ;
   x:=temp
```

Le principe de l'algorithme est de faire tourner ce vecteur `x`, et de permettre ainsi à chaque programme de faire successivement des opérations de la forme produit d'une matrice `(r,r)` par `x`. Les résultats sont accumulés dans les vecteurs `y` de chaque processeur. A la fin du programme, le vecteur `y` du processeur `i` contient $c_{\mathtt{ir}+1}, \ldots, c_{(\mathtt{i}+1)\mathtt{r}}$. Comme en (B), on fait usage d'une adresse `temp`. CQFD

9 Ordonnancement

Source. A. Darte et al, “Scheduling and automatic parallelization”, Birkhaüser (1999).

Un système de tâches est un graphe orienté $G = (V, E)$ muni d'une fonction de poids $w : V \to \mathbf{N}^\star$ sur les sommets, à valeurs dans les entiers naturels strictement positifs. Un horaire pour (V, E, w) est une fonction $\sigma : V \to \mathbf{N}^\star$ vérifiant :

$$\forall\,(u, v) \in E \;\; \sigma(u) + w(u) \leq \sigma(v) \ .$$

Interprétation : la tâche v ne peut commencer (au top $\sigma(v)$) que si toute tâche u dont elle dépend ($(u, v) \in E$) est terminée ($w(u)$ étant sa durée). Par défaut, on peut faire l'hypothèse que ces tâches sont exécutées l'une après l'autre sur un seul processeur. Mais tout l'intérêt de la formalisation viendra un peu plus loin lorsqu'on prendra en compte la possibilité d'une exécution sur plus d'un processeur.

(A) Montrer qu'un système de tâches (V, E, w) admet au moins un horaire si et seulement si (V, E) est un graphe acyclique orienté, i.e., si (V, E) n'a pas de cycle orienté.

Preuve. Soit $v_1, \ldots, v_k, v_1$ un cycle (i.e., $(v_i, v_{i+1}) \in E$ pour $i < k$ et $(v_k, v_1) \in E$), et supposons qu'il existe un horaire σ. On a a fortiori $\sigma(v_2) \leq \cdots \leq \sigma(v_k) \leq \sigma(v_1)$, ce qui contredit $\sigma(v_1) < \sigma(v_1) + w(v_1) \leq \sigma(v_2)$. Réciproquement, s'il n'y a pas de cycle, on peut faire un tri topologique, i.e., faire une liste des sommets de telle sorte que u apparaisse avant v dans l'énumération, pour toute arête (u, v) de G. (Pour réaliser un tri topologique, on prend une racine, i.e., un sommet u tel que $\nexists w \;\; (w, u) \in E$, et on supprime toutes les arêtes partant de u ; on prend ensuite une racine du graphe ainsi amputé, et ainsi de suite.) Soit $v_1, \ldots, v_n$ un tri topologique des sommets. Alors on pose $\sigma(v_1) = 0$ et $\sigma(v_{i+1}) = \sigma(v_i) + w(v_i)$ (le mieux que l'on puisse faire avec un seul processeur). CQFD

On fait désormais l'hypothèse que (V, E) est acyclique orienté. Une implantation sur un ensemble fini ou infini $\mathcal{P}$ de processeurs de cardinal p est donnée par un horaire et une fonction supplémentaire $\textsc{alloc} : V \to \mathcal{P}$ vérifiant :

$$(v_1 \neq v_2 \text{ et } \textsc{alloc}(v_1) = \textsc{alloc}(v_2)) \;\; \Rightarrow \;\; \left\{ \begin{array}{l} (\sigma(v_1) + w(v_1) \leq \sigma(v_2) \text{ ou} \\ \sigma(v_2) + w(v_2) \leq \sigma(v_1)) \ . \end{array} \right.$$

On dit que ALLOC convient pour σ et on écrit ALLOC $\models_p \sigma$, et on dit que σ est un horaire pour p processeurs. Etant donnés σ, p et ALLOC (qui reste implicite) tels que ALLOC $\models_p \sigma$, on note

$$MS(\sigma, p) = \max_{v \in V}(\sigma(v) + w(v)) - \min_{v \in V}(\sigma(v)) \; .$$

(En général on définit σ de sorte que $\min_{v \in V}(\sigma(v)) = 0$ – c'est évidemment possible par translation.) Pour $p \in \mathbf{N} \cup \{\infty\}$, on pose :

$$MS(p) = \min\{MS(\sigma, p) \mid \exists\, \textsc{alloc} \;\; \textsc{alloc} \models_p \sigma\} \; .$$

On rappelle qu'un chemin est une suite $\Phi = v_1, \ldots, v_n$ de sommets telle que $(v_i, v_{i+1}) \in E$ pour tout i ; on pose $w(\Phi) = w(v_1) + \cdots + w(v_n)$. On regarde d'abord le cas $p = \infty$.

(B1) Calculer $MS(\infty)$ et donner un horaire σ tel que $MS(\sigma, \infty) = MS(\infty)$. Evaluer la complexité de cette recherche.

(B2) Existe-t-il d'autres horaires optimaux (toujours avec $p = \infty$) ? Utilisent-ils toujours le nombre minimum de processeurs ? Pour ce dernier point, considérer l'exemple suivant :

$$\begin{array}{rcl} V & = & \{v_1, v_2, v_3, v_5, v_5, v_6, v_7, v_8\} \\ E & = & \{((v_1, v_3), (v_1, v_5), \\ & & (v_2, v_3), (v_2, v_4), (v_2, v_5), \\ & & (v_3, v_5), \\ & & (v_4, v_6), (v_4, v_7), \\ & & (v_5, v_7), \\ & & (v_6, v_8)\} \end{array}$$

PREUVE. (B1) Puisque les ressources sont illimitées, les seules contraintes sont celles données par E. Ceci conduit à poser

$$\sigma(v) = \max\{w(\Phi) - w(v) \mid \Phi \text{ part d'une racine et termine en } v\} \; .$$

On nomme cet horaire ASAP ("as soon as possible"). On peut calculer σ par récurrence sur un parcours préfixe du graphe :

$$\sigma(v) = \max\{\sigma(u) + w(u) \mid (u, v) \in E\} \; .$$

L'horaire σ est clairement optimal, et $MS(\infty) = \max_{v \in V}(\sigma(v) + w(v))$ est le poids maximum d'un chemin de G . La complexité est linéaire en la somme du nombre de sommets et d'arêtes de G (prix d'un parcours de G pour accumuler les informations sur σ).

(B2) On peut aussi démarrer le plus tard possible. Cela peut sembler surprenant à première vue, mais la surprise s'efface si on considère le graphe obtenu en renversant le sens des flèches, dont les racines sont les feuilles (i.e., les sommets u tels que $\nexists w \;\; (u, w) \in E$) du graphe G. Voici cet horaire, appelé ALAP ("as late as possible"), qui est donc également optimal :

$$\tau(v) = MS(\infty) - \max\{w(\Phi) - w(v) \mid \Phi \text{ part de } v \text{ et termine sur une feuille}\} .$$

Dans l'exemple de l'énoncé, les horaires ASAP (à gauche ci-dessous) et ALAP (à droite) utilisent 3 processeurs, celui du milieu n'en utilise que 2 (le nombre de processeurs est le nombre de colonnes) :

$$\begin{array}{lll} v_1 & v_2 & \\ v_3 & v_4 & v_5 \\ v_6 & v_7 & \\ v_8 & & \end{array} \qquad \begin{array}{ll} v_1 & v_2 \\ v_3 & v_4 \\ v_6 & v_5 \\ v_8 & v_7 \end{array} \qquad \begin{array}{lll} v_1 & & \\ v_1 & v_4 & \\ v_3 & v_6 & v_5 \\ v_8 & v_7 & \end{array}$$

CQFD

Le cas où le nombre de processeurs est fini est plus compliqué.

(C1) Analyser l'exemple suivant (les poids sont indiqués en indices supérieurs) :

$$\begin{array}{rcl} V & = & \{v_i^{K(p-1)} \mid 1 \leq i \leq p-1\} \cup \{v_p^1\} \\ & & \cup\{v_i^K \mid p+1 \leq i \leq 2p\} \cup \{v_{2p+1}^{K(p-1)}\} \\ E & = & \{(v_p, v_i) \mid p+1 \leq i \leq 2p\} \\ & & \cup\{(v_i, v_{2p+1}) \mid p+1 \leq i \leq 2p\} \end{array}$$

Décrire un horaire σ glouton (i.e. qui laisse à chaque top le moins possible de processeurs inactifs), et donner un horaire optimal. Calculer

$$MS(\sigma, p)/MS(p)$$

et discuter la valeur asymptotique de ce rapport.

(C2) Montrer, pour tout système de tâches, que la limite α obtenue en (C1) est une borne supérieure pour tout horaire glouton (au sens ci-dessus), i.e. : $MS(\sigma, p) \leq \alpha MS(p)$. Indication : construire un chemin Φ tel que :

$$(\mathbf{P_1}) \quad \mathit{Idle} \leq (p-1) \times w(\Phi) \;,$$

où *Idle* désigne le temps cumulé d'inactivité des processeurs.

Preuve. (C1) Les tableaux ci-dessous représentent le déroulement de l'horaire glouton, puis de l'horaire optimal :

de	à					
0	1	v_1	v_2	$\cdots$	v_{p-1}	v_p
1	$K+1$	v_1	v_2	$\cdots$	v_{p-1}	v_{p+1}
$\vdots$						
$K(p-3)+1$	$K(p-2)+1$	v_1	v_2	$\cdots$	v_{p-1}	v_{2p-2}
$K(p-2)+1$	$K(p-1)$	v_1	v_2	$\cdots$	v_{p-1}	v_{2p-1}
$K(p-1)$	$K(p-1)+1$	v_{2p}				v_{2p-1}
$K(p-1)+1$	Kp	v_{2p}				
Kp	$Kp+K(p-1)$	v_{2p+1}				

0	1					v_p
1	$K+1$	v_{p+1}	v_{p+2}	$\cdots$	v_{2p-1}	v_{2p}
$K+1$	$K+1+K(p-1)$	v_1	v_2	$\cdots$	v_{p-1}	v_{2p+1}

Les temps respectifs sont donc $K(2p-1)$ et $Kp+1$. On a

$$\frac{MS(\sigma,p)}{MS(p)} = \frac{K(2p-1)}{Kp+1} = (2p-1)\frac{K}{Kp+1} = (2p-1)\frac{1}{p+\frac{1}{K}} .$$

Donc $\lim_{K\to\infty} MS(\sigma,p)/MS(p) = 2-1/p$. La morale de cet exemple est que l'horaire glouton ne permet pas de reconnaître une situation où une tâche rapide libère un ensemble de tâches de durée moyenne et est par ailleurs accessible en même temps que des tâches de longue durée. Il peut être alors préférable de retarder ces tâches de longue durée pour paralléliser d'abord les tâches de moyenne durée.

(C2) Soit v_0 une tâche dont l'exécution s'achève au top $MS(\sigma,p)$. On définit t_0 comme suit. Si tous les processeurs sont actifs à chaque top $t < \sigma(v_0)$, alors on pose $t_0 = 0$. Sinon, t_0 est défini par :
- $t_0 < \sigma(v_0)$,
- l'un des processeurs est inactif au top t_0,
- pour tout top t tel que $t_0 < t < \sigma(v_0)$, tous les processeurs sont actifs au top t.

Puisque un processeur est inactif au top t_0, c'est qu'à cet instant aucune tâche qui peut être exécutée au regard des contraintes de précédence n'est en attente d'un processeur libre. Comme v_0 n'a pas commencé ($t_0 < \sigma(v_0)$), on en déduit qu'il existe une tâche v_1 en cours d'exécution au top t_0, et un chemin de v_1 à v_0 dans G. Puis on définit $t_1 < \sigma(v_1)$ et v_2 de la même manière, et ainsi de suite jusqu'à $t_m = 0$. Au total, les seuls intervalles durant lesquels une inactivité d'au moins un (et au plus $p-1$) processus est possible sont :

$$[\sigma(v_m), t_{m-1}] \quad \cdots \quad [\sigma(v_1), t_0] \quad [\sigma(v_0), MS(\sigma, p)]$$

A fortiori, cette inactivité d'au plus $p-1$ processeurs a lieu pendant les intervalles

$$[\sigma(v_m), \sigma(v_m) + w(v_m)] \quad \cdots \quad [\sigma(v_1), \sigma(v_1) + w(v_1)] \quad [\sigma(v_0), MS(\sigma, p)] \ ,$$

d'où l'inégalité de l'énoncé (avec $\Phi = v_m, \ldots, \ldots, v_1, \ldots v_0$). On peut maintenant terminer la preuve, grâce à cette autre inégalité (valable pour tout chemin) :

$$(\mathbf{P_2}) \quad MS(\sigma, p) \geq w(\Phi) \ .$$

Preuve de P_2. Puisque v_{i+1} dépend de v_i, on a $\sigma(v_i) + w(v_i) \leq \sigma(v_{i+1})$, donc a fortiori $\sigma(v_i) + w(v_i) \leq \sigma(v_{i+1}) + w(v_{i+1})$. Donc

$$\begin{array}{rcl} MS(\sigma, p) & \geq & \sigma(v_n) + w(v_n) \\ & \geq & \sigma(v_{n-1}) + w((v_{n-1})) + w(v_n) \\ & \vdots & \\ & \geq & \sigma(v_1) + w(\Phi) \geq w(\Phi) \ . \end{array}$$

Retour à (C2). On pose $Seq = \sum_{v \in V} w(v))$. On a :

$$\begin{array}{rcl} p \times MS(\sigma, p) & = & \mathit{Idle} + Seq \quad \text{(une porte est ouverte ou fermée)} \\ & \leq & \mathit{Idle} + (p \times MS(p)) \quad \text{(idem, pour un horaire optimal)} \\ & \leq & ((p-1) \times w(\Phi)) + (p \times MS(p)) \quad (P_1) \\ & \leq & ((p-1) \times MS(p)) + (p \times MS(p)) \quad (P_2) \\ & = & (2p-1) \times MS(p) \ . \end{array}$$

CQFD

10 Commutation de mots

Source. M. Harrison, “Introduction to Formal Language Theory”, Addison-Wesley (1978), théorèmes 1.3.3 et 1.3.4.

On se donne un alphabet Σ. La longueur d'un mot u est notée $|u|$. On écrit disons $u = w^3$ pour $u = www$ et on dit que u est puissance de w. Il s'agit de montrer :

(A1) Pour tous mots $u, v \in \Sigma^*$, il existe w tel que u et v sont puissances de w si et seulement si les mots infinis $u \cdots u \cdots$ et $v \cdots v \cdots$ contiennent un préfixe commun de longueur $|u| + |v| - pgcd(|u|, |v|)$. Et cette borne inférieure ne peut être améliorée.

(A2) Pour tous mots u, v, si $uv = vu$, alors u et v sont puissances d'un même mot w.

(A3) Pour tous u, v, le semi-groupe $\{u, v\}^+$ est libre si et seulement si $uv \neq vu$. (Un semi-groupe est un ensemble X muni d'une loi de composition associative notée multiplicativement ; il est libre si $(x_1 \cdots x_m = y_1 \cdots y_n \Rightarrow m = n, x_1 = y_1, \ldots, x_n = y_n)$, pour tous $x_1, \ldots, x_m, y_1, \ldots, y_n \in X$.)

Preuve. (A1) La partie “seulement si” est évidente, puisqu'alors les deux mots infinis coïncident avec $w \cdots w \cdots$. On pose $s = |u|$ et $t = |v|$. Pour voir que la borne ne peut être améliorée, on prend $u = a$ et $v = ab$: les deux mots infinis ont pour préfixe commun a dont la longueur est juste inférieure de 1 à la borne $1 + 2 - 1$.
La partie sérieuse est le “si”. Si $pgcd(s, t) = d > 1$, on se ramène au cas $pgcd(s, t) = 1$ en passant de l'alphabet Σ à l'alphabet dont les lettres sont les mots de longueur d. On peut supposer $t > s$, soit $t = ks + l$ avec $0 < l < s$. (Il peut être bon de s'essayer avec un exemple, comme $s = 7, t = 10$). On pose

$$u = a_0 \cdots a_{s-1} \quad v = b_0 \cdots b_{t-1} \ .$$

L'hypothèse de préfixe commun s'écrit

$$\begin{array}{ccccccccccccc}
a_0 & \cdots & a_{s-1} & a_0 & \cdots & a_0 & \cdots & a_{l-1} & a_l & \cdots & a_{s-1} & a_0 & \cdots & a_{(s-2+l \bmod l)} \\
b_0 & \cdots & b_{s-1} & b_s & \cdots & b_{ks} & \cdots & b_{ks+l-1} & b_0 & \cdots & b_{s-l-1} & b_{s-l} & \cdots & b_{s-2} \\
 & & & & & & & & a_0 & \cdots & a_{s-l-1} & a_{s-l} & \cdots & a_{s-2}
\end{array}$$

la dernière ligne étant un report du début de la première. On a donc $a_l = a_0$, $a_{l+1} = a_1$, et plus généralement (puisque $l = t \bmod s$) :

$$a_{(i+t \bmod s)} = a_i \quad (0 \leq i < s - 1).$$

Ceci permet déjà d'avoir $a_0 = a_l = \cdots = a_{(k'l \bmod s)}$, où la suite s'arrête au premier $k' < s-1$ tel que $(k'l \bmod s) = s-1$ s'il y en a, sinon $k' = s-2$. Dans ce dernier cas, on aura énuméré $s-1$ valeurs différentes $(jl \bmod s)$ $(0 \leq j < s-1)$ dans $\{0, \ldots, s-2\}$, i.e. on a $a_0 = a_1 = \cdots = a_{s-2}$ (on s'est servi ici de $pgcd(s,t) = 1$) ; en plus, en se plaçant en a_{s-1} à droite de a_l, on obtient que a_{s-1} est égal à l'un des a_j $(0 \leq j < s-1)$. Donc $a_0 = a_1 = \cdots = a_{s-2} = a_{s-1}$. Considérons maintenant l'autre cas : l'idée est de partir "dans l'autre sens". On a $a_0 = a_{(-l \bmod s)}$ en se plaçant en a_0 à droite de a_l, et on obtient successivement $a_0 = a_{(-l \bmod s)} = \cdots = a_{(-k''l \bmod s)}$ où $k'' = s - k' - 1$. Noter que l'on reste bien dans $\{0, \ldots, s-2\}$ puisque $s-1$ a déjà été atteint ($a_{(k'l \bmod s)} = a_{s-1}$ par hypothèse). Au total, on a (en omettant les "mod") :

$$a_{-k''l} = \cdots = a_{-l} = a_0 = a_l = \cdots = a_{k'l}$$

qui énumèrent $\{a_i \mid 0 \leq i < s\}$. Soit a cette valeur commune. Alors $u \cdots u \cdots = a \cdots a \cdots$, d'où $a = b_0 = \cdots = b_{t-1}$. Donc u et v sont tous deux puissances de a.

(A2) Remarquer que

$$vuu \cdots u \cdots = uvu \cdots u \cdots = uuv \cdots u \cdots = uu \cdots uv \cdots = uu \cdots u \cdots$$

puis que $v \cdots vu \cdots u \cdots = u \cdots u \cdots$.

(A3) Si le semi-groupe est libre, alors en particulier $uv \neq vu$. Réciproquement, si le semi-groupe n'est pas libre, on peut écrire une égalité entre puissances alternées de u et v de la forme

$$(i) \quad u^m v^{m'} \cdots v^n = v^p u^{p'} \cdots u^q \quad \text{ou} \quad u^m v^{m'} \cdots u^n = v^p u^{p'} \cdots v^q$$

(avec $m, n, p, q > 0$) en utilisant le cas échéant des annulations à gauche ($uu' = uu'' \Rightarrow u' = u''$) et à droite. On suppose que $|v| \leq |u|$. Il résulte de (i) que v est un préfixe de u. On écrit

$$(ii) \;\; u = v^r t_0 \quad (r > 0)$$

où v n'est pas préfixe de t_0. Alors le mot à gauche dans (i) commence par $v^r t_0$ tandis que le mot à droite commence par $v^p v^r t_0$. Par annulation à gauche, on obtient $t_0 \cdots = v^p t_0 \cdots$, et comme v n'est pas préfixe de t_0, on a $v = t_0 t_1$. Maintenant, par (i), $v = t_0 t_1$ est un suffixe de u, tandis que, par (ii), $t_1 t_0$ est un suffixe de u, ce qui implique que $t_0 t_1 = t_1 t_0$. Par (A2), il existe un mot w dont t_0

et t_1 sont des puissances, et donc aussi $v = t_0 t_1$ et $u = v^r t_0$. Donc en particulier $vu = uv$. On a bien montré par l'absurde que si $uv \neq vu$, alors $\{u, v\}^+$ est libre. CQFD

(B) Si $u = w^n$, on note $\rho(u)$ le mot de longueur minimum tel que u est une puissance de $\rho(u)$. Améliorer l'énoncé (A2) en montrant que si $uv = vu$, alors $\rho(u) = \rho(v)$.

PREUVE. Les mots $\rho(u)$ et $\rho(v)$ satisfont une équation de la forme $\rho(u)^m \, \rho(v)^n = \rho(v)^n \rho(u)^m$, donc $\{\rho(u), \rho(v)\}^+$ n'est pas libre, et, par (A3), on a $\rho(u)\rho(v) = \rho(v)\rho(u)$. Par (A2), $\rho(u)$ et $\rho(v)$ sont puissances d'un même w. Par minimalité, on a $\rho(u) = w = \rho(v)$. CQFD

11 Graphes admettant un 1-facteur

Source. F. Harary, “Graph Theory”, Addison-Wesley (1969), théorème 9.4.

Rappels. Un graphe est un couple $G = (V, E)$, $E \subseteq V \times V$. Deux arêtes sont dites incidentes si elles ont un sommet commun. Idem pour sommet incident à arête. Un circuit est une suite d'arêtes adjacentes toutes distinctes, notée $u_0 u_1 \cdots u_n$ $(u_i \in V, (u_i, u_{i+1}) \in E)$. Noter qu'il peut y avoir croisement dans un circuit, i.e. $u_i = u_j$ pour deux sommets de la suite. Si tous les sommets sont distincts (ce qui implique a fortiori que les arêtes sont distinctes), on dit qu'on a un chemin. La relation “être relié par un chemin (ou un circuit)” est une relation d'équivalence dont les classes d'équivalence sont appelées les composantes connexes du graphe (noter que de tout circuit on peut extraire un chemin, par suppression de boucles éventuelles). Une composante est dite paire (impaire) si elle est de cardinal pair (impair). Un 1-facteur de G, s'il en existe, est un graphe constitué d'arêtes de G indépendantes (i.e. non incidentes deux à deux), dont l'ensemble des sommets est V tout entier (le 1 dans 1-facteur vient de ce que ce sous-graphe est régulier d'ordre 1, i.e., chaque sommet n'est incident qu'à une arête du sous-graphe). En particulier $\sharp V$ (cardinal de V) est pair. Notation : $G - W$ (où $W \subseteq V$) est le graphe obtenu en enlevant les sommets qui sont dans W et avec eux toutes les arêtes qui leur sont incidentes. Il s'agit de montrer :

- pour tout graphe $G = (V, E)$, G a un 1-facteur si et seulement si il n'existe pas $W \subseteq V$ tel que $G - W$ ait plus de $\sharp W$ composantes impaires.

Pour la mise en jambes, constater que la condition du théorème implique que $\sharp V$ est pair (prendre $W = \emptyset$; G ne peut avoir que des composantes paires). N.B. : cette observation ne sert pas dans la preuve. La partie “seulement si” est facile. Soit $W \subseteq V$ et H une composante impaire de $G - W$. Le 1-facteur induit une bijection involutive entre une moitié de V et l'autre. Il y a au moins un sommet de H dont l'image par l'involution n'est pas dans H, sinon $\sharp H$ serait pair. Cette image est donc dans W (sinon, par connexité, elle devrait être dans H). On peut définir ainsi une fonction injective (par l'indépendance des arêtes du facteur) des composantes impaires de $G - W$ dans W, ce qui achève l'argument dans cette direction.

Pour l'autre direction, on introduit un attirail intrigant. On suppose que G n'a pas de 1-facteur, et on isole un sous-ensemble S d'arêtes indépendantes de G de cardinalité maximale. Puisque G n'a pas de 1-facteur, il y a au moins un sommet u_0 de G qui n'est incident à aucune arête de S. On note T l'ensemble $E - S$ (les autres arêtes de G). On s'intéresse aux circuits vers u_0 dont les arêtes alternent entre S et T que l'on appelle S-circuits ou T-circuits selon qu'ils commencent par une arête de S ou de T. Par le choix de u_0, tous ces circuits se terminent par une arête de T (et ne passent d'une fois par u_0). On introduit la terminologie suivante pour les sommets $u \neq u_0$ de G :

- u est un S-point (T-point) s'il existe un S-circuit (T-circuit) de u à u_0.
- On dit que u est un ST-point s'il est à la fois un S-point et un T-point, et que u est un S^*-point (T^*-point) s'il est un S-point mais non un T-point (un T-point mais non un S-point).

On observe d'abord que les S-circuits et les T-circuits sont "presque" des chemins. A cause de l'alternance et de l'indépendance, il peut seulement y avoir une forme dégénérée de croisement : $uv \cdots v'uw \cdots u_0$, avec $(u,v) \in T$. La propriété suivante sera utile :

$$\textbf{(A)} \quad u \;\; S\text{-point et } (u,v) \in E \;\Rightarrow\; v \;\; T\text{-point} \quad (v \neq u_0)\ .$$

Si $(u,v) \in S$, il suffit de couper la première arête d'un S-circuit à partir de u. Si $(u,v) \in T$, il est facile de voir qu'on peut toujours ajouter une arête de T au début d'un S-circuit et obtenir ainsi un T-circuit. On a aussi la propriété suivante :

$$\textbf{(B)} \;\; \forall\, u \in T^* \;\; \exists\, v \;\; (u,v) \in S\ .$$

Pour le voir, on prend un T-chemin partant de u (en coupant d'éventuelles boucles initiales, cf. ci-dessus). Soit S_1 (T_1) l'ensemble des arêtes de S (T) le long de ce chemin. Si (B) est faux pour u, alors $T_1 \cup (S \setminus S_1)$ est un ensemble d'arêtes indépendantes, ce qui contredit l'hypothèse de maximalité, car $\sharp T_1 = \sharp S_1 + 1$. Donc (B) est vrai ; il est facile de voir que le v de l'énoncé est unique, et est un S-point, ce qui définit une injection des T^*-points dans l'ensemble des S-points.

On prend alors pour W l'ensemble des T^*-points, et l'on va montrer entre autres que toutes les composantes connexes de $G - W$ contenant des S-points sont impaires.

(C) Chaque S^*-point constitue à lui seul une composante connexe (sommet isolé) ; chaque ST-point est membre d'une composante constituée exclusivement de ST-points (en plus de u_0).

Soit u un S^*-point et $(u,v) \in E$. Par (A), v est un T-point. Supposons que v n'est pas un T^*-point. C'est alors un ST-point ; mais alors, par (A) encore, u est un T-point. Contradiction. Donc u est un sommet isolé de $G - W$, puisqu'il n'est adjacent dans G qu'à des T^*-points. Soit u un ST-point et $(u,v) \in E$. Par (A),

v est un T-point ou $v = u_0$. Donc tout sommet $v \neq u_0$ adjacent à un ST-point dans $G - W$ est lui-même un ST-point.

On appelle H_0 la composante connexe de u_0, et H_1 une composante connexe quelconque formée de ST-points. On montre que H_1 possède un sommet u_1 relié à un T^*-point v_1 par une arête de S. Soit u un sommet de H_1, et soit un S-circuit ou T-circuit de u_0 à u. Sur ce circuit, soit u_1 le premier sommet à partir de u_0 qui appartient à H_1, et soit v_1 son prédécesseur, qui est un T^*-point puisque hors de la composante et adjacent à un sommet de la composante. Donc la première arête sur le circuit de v_1 à u_0 est dans T, ce qui implique par alternance $(v_1, u_1) \in S$. On note P_0 le circuit de u_0 à u_1, qui est par construction hors de H_1 jusqu'à v_1. On montre ensuite :

(D) Tout sommet u de H_1 (H_0) est relié à u_1 (u_0) par un circuit alternant contenu dans H_1 et se terminant (en u) par une arête de S.

On montre d'abord que l'existence d'un circuit P_1 alternant dans H_1 de u_1 à u implique l'existence d'un circuit alternant dans H_1 de u_1 à u se terminant de plus par une arête de S. Puisque u est un ST-point, on peut construire un S-circuit P_2 de u_0 à u, qui quelque part rentre dans H_1 par une arête $(v', u') \in S$ avec $v' \in T^*$ (cf. ci-dessus). Au premier sommet où le circuit P_1 rencontre P_2, on peut continuer sur P_2 dans le sens dicté par l'alternance. Ce sens ne peut être celui vers v', car en concaténant avec P_0, on aurait depuis u_0 un S-circuit vers v' via u_1 et u'. Or v' n'est pas un S-point. Donc le circuit alternant fabriqué à l'aide de P_1 et P_2 se termine en u, par une arête de S par construction de P_2. Un argument semblable (moins le raccordement par P_0) s'applique à H_0, u_0 et n'importe quel sommet u de H_0.
On démontre maintenant l'existence des circuits P_1, par récurrence sur la longueur des circuits à partir de u_1 (u_0) dans H_1 (H_0). Si le circuit est $u_1 \cdots vu$, par récurrence, on peut supposer qu'il existe un circuit alternant P pour v. Si $(v, u) \in S$, par l'hypothèse d'indépendance sur S, la dernière arête de P est forcément dans T. On obtient un circuit alternant soit en ajoutant (v, u) à P, soit en coupant P après cette arête. Si $(v, u) \in T$, alors on utilise la propriété déjà prouvée que l'existence de P implique celle d'un circuit se terminant par S, auquel on peut ajouter (v, u). Ceci achève la preuve de (D).
On ne retiendra de (D) que le fait que tout sommet de H_1 (H_0) autre que u_1 (u_0) est apparié à un autre sommet de H_1 (H_0) par une arête de S. On en déduit que H_1 (H_0) est impaire, et que u_1 est le seul sommet de H_1 raccordé par une arête de S à un T^*-point, tandis qu'aucun sommet de H_0 n'a cette propriété.

On peut maintenant récolter les dividendes. Par (C) et les conséquences de (D) que l'on vient d'énoncer, l'injection conséquence de (B) peut être vue comme une injection de W dans les composantes impaires de $G - W$ différentes de H_0, et le nombre de composantes impaires excède donc $\sharp W$. CQFD

12 Algorithme du simplexe

Source. P. Morris, “Introduction to Game Theory”, Springer (1994), chapitres 3 et 4.

On considère des matrices (t_{ij}) à $m+1$ lignes et $n+1$ colonnes à coefficients réels, avec un étiquetage de la forme x_p $(-x_q)$ pour chaque colonne (ligne) sauf la dernière, toutes ces variables étant distinctes. On les appelle variables de colonne et de ligne respectivement. On complète cet étiquetage en mettant -1 au-dessus de la dernière colonne et un nom de fonction f à droite de la dernière ligne. On note souvent :

$$t_{(m+1)j} = c_j \ \ (j \leq n) \quad t_{i(n+1)} = b_i \ \ (i \leq m) \quad t_{(m+1)(n+1)} = d$$

et de même c'_j, b'_i, d' pour une matrice (t'_{ij}). Graphiquement :

$$\begin{array}{cccc|c} \cdots & x_p & \cdots & -1 & \\ \hline & \vdots & & \vdots & \\ \cdots & t_{ij} & \cdots & b_i & -x_q \\ & \vdots & & \vdots & \\ \cdots & c_j & \cdots & d & f \end{array}$$

Une telle matrice étiquetée, ou tableau, est la transcription d’un problème d’optimisation. En supposant (ce que l’on peut toujours faire modulo renommage) que les variables de colonne sont $x_1, \ldots, x_n$ et les variables de ligne $x_{n+1}, \ldots, x_{n+m}$, le problème est : maximiser $f(x_1, \ldots, x_n) = \sum_{j=1}^{n} c_j x_j - d$ avec les contraintes $x_i \geq 0$ $(i \leq n)$ et $\sum_{j=1}^{n} t_{ij} x_j \leq b_i$ pour chaque $i \leq m$, ou, ce qui est équivalent, les contraintes $x_i \geq 0$ $(i \leq n+m)$ et $\sum_{j=1}^{n} t_{ij} x_j - b_i = -x_{n+i}$ (d’où l’étiquette -1 au-dessus de la colonne des b_i). Dans la suite, les contraintes $x \geq 0$ seront toujours implicites. Un vecteur satisfaisant ces contraintes est dit convenable. On dit que le tableau est convenable si $(\forall\, i \leq m \ \ b_i \geq 0)$. Dire que le tableau est convenable, c’est dire que le vecteur $(0, \ldots, 0, b_1, \ldots, b_m)$ (noté $\vec{0}$ pour faire bref) est convenable. Si le tableau est convenable, on dit que $-d = f(\vec{0})$ est la valeur du tableau.

(A) Donner une condition suffisante pour qu’un tableau convenable fournisse avec le vecteur $\vec{0}$ une solution au problème d’optimisation associé. Donner une condition suffisante pour que le problème associé à un tableau convenable ne soit pas borné (i.e., que f puisse prendre des valeurs arbitrairement grandes).

Preuve. La première condition est que tous les c_j soient ≤ 0. En effet, f sera alors toujours plus petit que $-d$, atteint en $\vec{0}$. La seconde condition est qu'il existe une colonne l parmi les n premières telle que $c_l > 0$ et $t_{il} \leq 0$ pour tout $i \leq m$. En effet, en posant $x = 0$ pour toutes les variables de colonne, sauf pour la variable x_p étiquetant la colonne l, pour laquelle on prend un nombre positif arbitrairement grand, on obtient un vecteur convenable : en effet, chaque contrainte s'écrit $-x_q = t_{il}x_p - b_i$ (si x_q est létiquette de la i-ème ligne), ce qui donne bien une valeur ≥ 0 à x_q puisque $t_{il} \leq 0$ par hypothèse et $b_i \geq 0$ (le tableau étant convenable). Et on a $f = c_l x_p - d$ arbitrairement grand. CQFD

Le but du jeu est maintenant d'intervertir une variable de colonne et une variable de ligne (pivot), pour passer d'un tableau convenable à un tableau convenable de valeur plus grande ou égale. L'opération de pivot est réalisable sur tout tableau autour de toute entrée $t_{kl} \neq 0$, comme suit (à droite ci-dessous) :

$$\begin{array}{ccccc|c} \cdots & x_p & \cdots & & \cdots & \\ \hline & \vdots & & \vdots & & \\ \cdots & t_{il} & \cdots & t_{ij} & \cdots & \\ & \vdots & & \vdots & & \\ \cdots & t_{kl} & \cdots & t_{kj} & \cdots & x_q \\ & \vdots & & \vdots & & \end{array} \qquad \begin{array}{ccccc|c} \cdots & x_q & \cdots & & \cdots & \\ \hline & \vdots & & \vdots & & \\ \cdots & -t_{il}/t_{kl} & \cdots & t_{ij} - (t_{il}t_{kj}/t_{kl}) & \cdots & \\ & \vdots & & \vdots & & \\ \cdots & 1/t_{kl} & \cdots & t_{kj}/t_{kl} & \cdots & x_p \\ & \vdots & & \vdots & & \end{array}$$

L'algorithme du simplexe :

1. Si $(\forall j \leq n \;\; c_j \leq 0)$, alors SUCCES. Sinon choisir $c_l > 0$.
2. Si $(\forall i \leq m \;\; t_{il} \leq 0)$, alors ECHEC. Sinon choisir $t_{kl} > 0$ tel que $b_k/t_{kl} = min\{b_i/t_{il} \mid i \leq m \text{ et } t_{il} > 0\}$.
3. Effectuer le pivot autour de t_{kl}, puis retourner à l'étape (1).

(B) Montrer que chaque étape (3) de l'algorithme du simplexe transforme un tableau convenable en un tableau (t'_{ij}) convenable, et que $d' \leq d$.

Preuve. Le choix de k rendant b_k/t_{kl} minimum garantit que la dernière colonne a ses m premières entrées ≥ 0, i.e., que (t'_{ij}) est convenable. Et on a $d' = d - (c_l b_k/t_{kl})$, avec

$$\begin{array}{ll} t_{kl} > 0 \text{ et } c_l > 0 & \text{(étapes (1) et (2) de l'algorithme)} \\ b_k \geq 0 & \text{(tableau convenable) .} \end{array}$$

CQFD

(C) L'algorithme du simplexe termine-t-il toujours ? Permet-il de décider si le problème d'optimisation sous-jacent a une solution et de la calculer ?

Preuve. On dit qu'une boucle de l'algorithme est croissante si elle diminue strictement l'entrée $(m+1),(n+1)$ du tableau. Supposons que l'algorithme ne termine pas, et que de plus il effectue un nombre infini de boucles croissantes. Il y a alors un nombre infini de valeurs différentes pour l'entrée d : or cette valeur est seulement déterminée par la partition de l'ensemble des variables en variables de colonne et variables de ligne (chaque opération de pivot effectue un changement de variable, ce qui rend f invariant en tant que fonction de l'ensemble des variables, et on a $-d = f(\vec{0}, \dots, x_q, \dots))$. Comme il n'y a qu'un nombre fini de partitions, il y a une contradiction. Donc la seule possibilité de non terminaison est qu'à partir d'un moment l'algorithme n'effectue plus que des boucles non croissantes, ce qui n'est pas possible si l'on impose une discipline de choix des colonnes puis des lignes dans l'algorithme (voir question (G)). Donc l'algorithme termine. S'il termine sur ECHEC, on a vu en (A) que le problème n'a pas de solution (non borné). S'il termine sur SUCCES, alors, par (A) aussi, le vecteur $\vec{0}$ sur le dernier tableau fournit une valeur maximale pour f. Mais encore faut-il pouvoir commencer l'algorithme : le tableau initial, traduction verbatim du problème d'optimisation, n'est pas nécessairement convenable. Voir question (H). CQFD

(D) Résumé. Montrer qu'au total on sait décider des trois situations exclusives et exhaustives suivantes

1. Le probème n'a pas de vecteur convenable.
2. Le problème admet au moins un vecteur convenable mais n'est pas borné.
3. Le problème admet une solution.

Preuve. Par (H), on sait décider de l'existence d'un vecteur convenable, et s'il en existe un, on sait produire au moins un tableau convenable. Par (C) et (G) on sait que l'algorithme du simplexe (qui peut partir de ce tableau) termine. Les cas (2) et (3) ci-dessus correspondent aux deux façons pour cet algorithme de terminer. CQFD

(E) Un tableau (t_{ij}) tel que ci-dessus est associé non seulement à un problème de maximisation, mais aussi à un problème de minimisation (modulo un autre étiquetage). Décrire ce problème, et montrer que les deux problèmes sont équivalents en ce sens que l'un a une solution (une valeur maximum pour f) si et seulement si l'autre a une solution (une valeur minimum pour la fonction g à minimiser), et que les deux valeurs sont égales.

Preuve. Comme plus haut, on convient que les variables de colonne sont $x_1, \ldots, x_n$ et les variables de ligne $x_{n+1}, \ldots, x_{n+m}$. Rappelons que le problème est : maximiser $\sum_{j=1}^{n} c_j x_j - d$ avec les contraintes $\sum_{j=1}^{n} t_{ij} x_j \leq b_i$. Le problème associé est : minimiser $g = \sum_{i=1}^{m} b_i y_i - d$ avec les contraintes $\sum_{i=1}^{m} t_{ij} y_i \geq c_j$. On représente ces deux problèmes avec la *même* matrice doublement étiquetée :

$$\begin{array}{c|cccc|c}
 & \cdots & x_j & \cdots & -1 & \\
\hline
 & & \vdots & & \vdots & \\
y_i & \cdots & t_{ij} & \cdots & b_i & -x_{n+i} \\
 & & \vdots & & \vdots & \\
-1 & \cdots & c_j & \cdots & d & f \\
\hline
 & \cdots & y_{m+j} & \cdots & g &
\end{array}$$

Et on constate que les deux parties de (A) s'interchangent : l'hypothèse de tableau convenable (fournissant un vecteur convenable pour le problème de maximisation) devient celle de solution et celle de solution devient celle de tableau convenable vis-à-vis du problème de minimisation. Ce qui fait qu'un tableau SUCCES pour l'algorithme ci-dessus s'interprète aussi comme fournissant une valeur minimale pour g, égale à la valeur maximale pour f. CQFD

(F) Application. Soit (M_{ij}) est une matrice $m \times n$ à coefficients réels, interprétée comme une matrice de gains. Deux joueurs appelés joueur "ligne" et joueur "colonne" ont respectivement m et n stratégies de jeux. Le gain de la partie est M_{ij} pour le joueur "ligne" et $-M_{ij}$ pour le joueur "colonne" lorsque ces joueurs jouent leur stratégie i et j, respectivement. Les joueurs peuvent aussi choisir une distribution de probabilités sur leurs stratégies. Si $p_1, \ldots, p_m$ est une distribution de probabilités sur les lignes, alors $min\{\sum_{i=1}^{m} p_i M_{ij} \mid 1 \leq j \leq n\}$, est le gain minimal que le joueur "ligne" peut espérer pour cette distribution. Il cherche donc à maximiser ce *min* sur l'ensemble des distributions de probabilités. Montrer que ce problème ainsi que le problème symétrique de l'autre joueur (qui est de minimiser les gains de l'autre) ont une solution et que les deux valeurs de gain obtenues (le max des min et le min des max) sont égales (cette propriété est appelée théorème du minimax).

Preuve. Le problème de maximisation peut s'énoncer comme : trouver v maximal tel que $v \leq \sum_{i=1}^{m} p_i M_{ij}$ pour chaque j et chaque vecteur de p_i tel que $p_i \geq 0$ pour tout i et $\sum_{i=1}^{m} p_i = 1$. Pour faire rentrer ces deux problèmes dans le moule ci-dessus, on opère un changement de variable qui permet d'exprimer la valeur v à maximiser comme une fonction des autres variables. Pour ce faire on pose

$y_i = p_i/v$ pour tous les i. Le problème devient : minimiser $\sum_{i=1}^m y_i$ $(= 1/v)$ avec les contraintes $\sum_{i=1}^m M_{ij} y_i \geq 1$. (N.B. : on peut garantir $v > 0$ an ajoutant le cas échéant une même constante assez grande à toutes les entrées de la matrice, ce qui ne change rien à l'analyse – la constante peut même être retranchée à la fin.) Sous forme de tableau :

$$\begin{array}{c|cccc|c}
 & \cdots & x_j & \cdots & -1 & \\
\hline
 & & \vdots & & \vdots & \\
y_i & \cdots & t_{ij} & \cdots & 1 & -x_{n+i} \\
 & & \vdots & & \vdots & \\
-1 & \cdots & 1 & \cdots & 0 & f \\
\hline
 & \cdots & y_{m+j} & \cdots & g &
\end{array}$$

Le problème associé se lit : maximiser $\sum_{j=1}^n x_j$ avec les contraintes $\sum_{j=1}^n M_{ij} x_j \leq 1$, ce qui, en effectuant un changement de variable en sens inverse en posant $q_j = x_j v$ devient : trouver v minimum tel que $\sum q_j = 1$ et $v \geq \sum M_{ij} q_j$, soit trouver le min des max, ce qui est bien le meilleur score que puisse espérer le joueur colonne. On note enfin que l'algorithme peut bien démarrer (le tableau est convenable) et ne peut se terminer sur ECHEC car $\sum_{i=1}^m y_i$ est bornée inférieurement (par 0). CQFD

(G) On suppose que l'ensemble des variables de colonne et de ligne est totalement ordonné (par exemple $x_p < x_q$ si et seulement si $p < q$). On spécifie davantage l'algorithme du simplexe comme suit. A l'étape (1), on choisit une colonne l étiquetée par la plus petite variable parmi celles étiquetant des colonnes j telles que $c_j > 0$ (règle de Bland (1)). A l'étape (2), on choisit aussi la plus petite variable parmi l'ensemble de celles étiquetant des lignes satisfaisant les conditions (règle de Bland (2)). Montrer qu'ainsi précisé l'algorithme du simplexe termine toujours.

PREUVE. Par (C), on doit seulement considérer le cas où l'algorithme ne change plus l'entrée d. On a $d = d'$ si et seulement si $b_k c_l = 0$ si et seulement si $b_k = 0$. On parle alors de pivot dégénéré. On a la propriété suivante (facile) :

(P) L'exécution d'un pivot dégénéré laisse la colonne de droite (étiquetée par -1) invariante.

Supposons donc que l'on ait une suite infinie de pivots dégénérés, ce qui revient à dire qu'il y a une suite de tableaux $T^0, \ldots, T^k$, appelée cycle, telle que T^k ait le même ensemble de variables de colonne que T^0. On définit

$$x_t = max\ C \quad \text{où } C = \{x_i \mid x_i \text{ devient une variable de ligne au cours du cycle}\}\ .$$

Il existe donc v tel que x_t est variable de colonne dans T^v et variable de ligne dans T^{v+1}. Puisque le cycle repasse par T^v, la variable x_t doit redevenir variable de colonne. Il existe donc un tableau T^u tel que le passage de T^u à T^{u+1} fait pivoter x_t (variable de ligne) et une variable x_s (de colonne). Comme $x_s \in C$ et $s \neq t$, on a $s < t$ par le choix de t. On utilise les notations suivantes (T^u à gauche, T^v à droite) :

$$\begin{array}{ccccc|c}
\cdots & x_s & \cdots & -1 & & \\
\hline
 & \vdots & & \vdots & & \\
\cdots & a_i & \cdots & b_i & & x_i \\
 & \vdots & & \vdots & & \\
\cdots & c_s & \cdots & d & & f
\end{array}
\qquad
\begin{array}{cccc|c}
\cdots & x_r & \cdots & -1 & \\
\hline
 & \vdots & & \vdots & \\
 & \vdots & & b_i & \\
 & \vdots & & \vdots & \\
\cdots & c_r^* & \cdots & d & f
\end{array}$$

On considère le vecteur $\vec{v}$ obtenu en posant $x_s = 1$ et $x_r = 0$ pour toutes les autres variables de colonne x_r de T^u, et donc $x_i = a_i - b_i$ pour les variables de ligne. Le calcul de f à partir de T^u donne $f(\vec{v}) = c_s - d$, et l'on a $c_s > 0$ (algorithme du simplexe, cas (1)). A partir de T^v, on lit $f(\vec{v}) = \sum c_i^*(b_i - a_i) + c_s^* - d$, ou $f(\vec{v}) = \sum c_i^*(b_i - a_i) - d$ (selon que x_s est ou non variable de colonne dans T^v), où la sommation porte sur les i tels que x_i est variable de ligne dans T^u et variable de colonne dans T^v. Pour ces i, on a donc

$$\begin{array}{ll}
i \in C & \text{(donc } i \leq t) \\
b_i = 0 & \text{(propriété (P))}\ .
\end{array}$$

On peut donc supprimer les b_i, et on a :

$$-\sum c_i^* a_i = c_s \quad \text{ou} \quad -\sum c_i^* a_i = c_s - c_s^*\ .$$

Si x_s est variable de colonne dans T^v, on ne peut avoir $c_s^* > 0$ (règle de Bland (1) appliquée dans T^v). On a donc toujours $\sum c_i^* a_i < 0$, et a fortiori $c_r^* a_r < 0$ pour l'un des termes de cette somme. On a $c_t^* a_t > 0$, car

$$\begin{array}{ll}
a_t > 0 & \text{(pivot dans } T^u) \\
c_t^* > 0 & \text{(colonne pivotée dans } T^v)\ .
\end{array}$$

Donc $r \neq t$. Comme on a observé plus haut que $r \leq t$, on a donc $r < t$. On termine l'argument comme suit :

$$\begin{array}{ll} a_r > 0 & \text{est exclu par la règle de Bland (2) appliquée dans } T^u \\ c_r^* > 0 & \text{est exclu par la règle de Bland (1) appliquée dans } T^v \;. \end{array}$$

Donc $a_r \leq 0$ et $c_r^* \leq 0$, ce qui implique $c_r^* a_r \geq 0$: contradiction. CQFD

(H) L'objectif est de montrer que si le problème admet un vecteur convenable, alors on peut construire un tableau convenable pour le problème (modulo changement de variables), et de trouver au passage un algorithme qui décide de l'existence d'au moins un tel vecteur. Indication : ajouter une colonne étiquetée par une nouvelle variable de colonne u et d'entrées toutes égales à -1, et mettre sur la dernière ligne des 0 partout sauf sous la colonne étiquetée u, i.e., on change la fonction à maximiser, qui est maintenant $-u$.

Preuve. Le nouveau problème est lié au problème de départ par la propriété suivante, qui est facile à établir.

(Q) Les contraintes du nouveau problème plus la contrainte $u = 0$ sont équivalentes, en tant que systèmes d'équations, aux contraintes du problème de départ. En particulier, le problème de départ admet au moins un vecteur convenable si et seulement si le nouveau problème a un vecteur convenable avec $u = 0$, qui est alors en fait une solution, 0 étant nécessairement maximal pour la fonction $-u$.

On peut supposer que l'un au moins des b_i est < 0 (sinon l'initialisation peut être faite d'emblée avec le tableau initial). On effectue un pivot qui échange la colonne u et une ligne ayant un b_i minimal :

$$\begin{array}{ccc|c} \cdots & u & -1 & \\ \hline & \vdots & & \\ \cdots & -1 & b_j & \\ & \vdots & & \\ \cdots & -1 & b_i & x_{n+i} \\ & \vdots & & \end{array} \qquad \begin{array}{ccc|c} \cdots & x_{n+i} & -1 & \\ \hline & \vdots & & \\ \cdots & -1 & b_j - b_i & \\ & \vdots & & \\ \cdots & -1 & -b_i & u \\ & \vdots & & \end{array}$$

On peut appliquer l'algorithme du simplexe au tableau ainsi obtenu, car il est convenable. L'algorithme ne peut se terminer sur ECHEC car la fonction $-u$ à optimiser est bornée. Il s'arrête donc. S'il s'arrête sur une valeur $d \neq 0$, c'est que le problème de départ n'a pas de vecteur convenable (par (Q)). S'il s'arrête avec $d = 0$, on distingue deux cas.

1. Si u est une variable de colonne du tableau d'arrivée, il suffit de retirer la colonne u et l'on obtient par la propriété (Q) un ensemble de m inéquations décrivant les contraintes du problème de départ. On obtient alors un tableau convenable pour ce problème en remettant en dernière ligne la fonction initiale à maximiser (modulo changement de variables).
2. Si u est une variable de ligne, étiquetant la ligne i, alors $b_i = 0$ (c'est la valeur de u) et au moins un t_{ij} est non nul (une ligne dont toutes les entrées sont 0 ne peut être introduite par l'algorithme, et aurait donc du être là dés le départ, ce qui n'est pas le cas). On peut donc effectuer un pivot dégénéré, qui ne modifie pas la colonne de droite (par la propriété (P)). On est alors ramené au cas (1).

CQFD

En résumé la procédure d'initialisation est la suivante :

1. Si le tableau initial est convenable, alors SUCCES. Sinon, ajouter la colonne u et changer la dernière ligne comme ci-dessus.
2. Effectuer le pivot initial entre la colonne u et une ligne ayant un b_i minimal.
3. Exécuter l'algorithme du simplexe. Si $d \neq 0$ dans le tableau final, alors ECHEC. Sinon SUCCES.
4. Si u est variable de colonne, supprimer la colonne, et restaurer la fonction initiale à maximiser.
5. Si u est variable de ligne, effectuer un pivot avec cette ligne, et aller en (4).

13 Arbres rouge et noir

Source. Th. Cormen, C Leiserson et R. Rivest, “Introduction à l’Algorithmique”, Dunod (1994), chapitres 13 et 14.

Arbres binaires de recherche. On considère les termes construits à l’aide d’un symbole Ω d’arité 0 et pour chaque entier naturel n un opérateur n d’arité 2. Par exemple, $3(5(\Omega,\Omega),6(4(\Omega,\Omega),\Omega))$ est un terme, qui représente l’arbre de sommet 3, dont le fils gauche est la feuille 5 et et dont le fils droit 6 n’a pas de fils droit et a la feuille 4 pour fils gauche. On formalise le prédicat $\in(k,s)$ (“le nombre k apparaît dans s”) comme suit :

$$\frac{}{\in(k,k(s,t))} \qquad \frac{\in(k,s)}{\in(k,n(s,t))} \qquad \frac{\in(k,t)}{\in(k,n(s,t))}$$

(noter le non-déterminisme : pour vérifier le prédicat, on ne sait pas a priori quelle règle appliquer). Un peu de formalisme général pour la manipulation des arbres. Une occurrence u est l’adresse d’un noeud (ou du sous-arbre sous ce noeud), donnée sous forme d’un mot sur l’alphabet $\{1,2\}$, s/u est le sous-arbre, $s(u)$ est l’étiquette du noeud (n ou Ω), $s[t/u]$ désigne le terme obtenu en remplaçant dans s le sous-arbre s/u par t. Enfin, on note $s[k/_\circ u]$ le résultat du remplacement dans s de $s/u = n(t_1,t_2)$ par $k(t_1,t_2)$. Par exemple, si $s = 3(5(\Omega,\Omega),6(4(\Omega,\Omega),\Omega))$, alors $s(\epsilon) = 3$ (ϵ désigne le mot vide), $s/21 = 4(\Omega,\Omega)$, $s[\Omega/2] = 3(5(\Omega,\Omega),\Omega)$, $s[2/_\circ 1] = 3(2(\Omega,\Omega),6(4(\Omega,\Omega),\Omega))$. On dit qu’un terme s est un arbre binaire de recherche si :

$$\forall u\ \ (s/u = k(t_1,t_2) \Rightarrow (\forall x\ \ (\in(x,t_1)) \Rightarrow x \le k) \text{ et } (\forall y\ \ (\in(y,t_2)) \Rightarrow k \le y))$$

(dans ce qui suit, on suppose en fait $<$, i.e. tous les nombres rangés dans l’arbre sont distincts).

Quelques algorithmes : le prédicat $\in$ restreint aux arbres de recherche binaire est maintenant déterministe, et peut s’écrire sous la forme suivante :

$$\frac{k = n}{[\in(k,n(s,t))] \ \rightarrow\ \text{SUCCES}} \qquad \frac{}{[\in(k,\Omega)] \ \rightarrow\ \text{ECHEC}}$$

$$\frac{k < n}{[\in(k,n(s,t))] \ \rightarrow \in(k,s)} \qquad \frac{k > n}{[\in(k,n(s,t))] \ \rightarrow \in(k,t)}$$

Pour un arbre binaire de recherche, on vérifie sans peine que $\in (k, s)$ est prouvable selon les règles données plus haut si et seulement si $[\in (k, s)] \to^* \text{SUCCES}$.

(A) Ecrire dans ce style les procédures de recherche du minimum, du successeur, d'insertion et de suppression de noeuds. Prouver la correction des algorithmes de successeur et de suppression.

PREUVE. Pour le minimum :

$$\frac{t/u1 \neq \Omega}{u \ \to \ u1} \qquad \frac{t/u1 = \Omega}{u \ \to !u}$$

et l'on a $min(t) = t(u)$ si et seulement si $\epsilon \to^* !u$. Pour le successeur :

$$\frac{t/u2 \neq \Omega}{u \ \to \ (u2, \downarrow)} \qquad \frac{t/v1 \neq \Omega}{(v, \downarrow) \ \to \ (v1, \downarrow)} \qquad \frac{t/v1 = \Omega}{(v, \downarrow) \ \to !v}$$

$$\frac{t/u2 = \Omega}{u \ \to \ (u, \uparrow)} \qquad \frac{v = w1}{(v, \uparrow) \ \to !w} \qquad \frac{v = w2}{(v, \uparrow) \ \to \ (w, \uparrow)}$$

Dans ces règles, $\downarrow$ ($\uparrow$) indique une recherche descendante (ascendante). La recherche descendante est celle du minimum du sous-arbre $t/u2$ dans le cas où $t/u2 \neq \omega$. Si au contraire $t/u2 = \Omega$, le successeur est à chercher plus haut (d'où le $\uparrow$). Pour voir que l'algorithme est correct, i.e., que l'on a $succ(t(u)) = t(v)$ si et seulement si $u \to^* !v$, on remarque que chaque application de la dernière règle maintient l'invariant que u est maximum dans l'arbre courant (i.e., le sous-arbre indiqué par le $(v, \uparrow)$ courant). Par ailleurs, la propriété (P) suivante des arbres binaires de recherche, dont la preuve est immédiate, garantit que le successeur n'est pas non plus à chercher plus haut :

(P) Si u, v, w sont tels que w est un préfixe de v et v est un préfixe de u (i.e., w est plus haut que v dans l'arbre et de même pour v et u), alors $t(u)$ et $t(v)$ sont tous deux soit plus petits soit plus grands que $t(w)$ (qui ne peut donc s'intercaler entre $t(u)$ et $t(v)$).

Pour l'insertion, on adapte simplement l'algorithme de test d'appartenance :

$$\frac{k < t(u) \quad t/u1 \neq \Omega}{u \ \rightarrow \ u1} \qquad \frac{k < t(u) \quad t/u1 = \Omega}{u \ \rightarrow !u1}$$

$$\frac{k > t(u) \quad t/u2 \neq \Omega}{u \ \rightarrow \ u2} \qquad \frac{k > t(u) \quad t/u2 = \Omega}{u \ \rightarrow !u2}$$

et si $\epsilon \rightarrow^* !u$, alors u est l'occurrence où insérer k : le nouvel arbre sera $t' = t[k(\Omega, \Omega)/u]$. Pour la suppression, on n'utilise pas (directement) de réécriture ; ci-dessous, $suppr(u, t)$ désigne un arbre qui est "t dans lequel on a supprimé le noeud u".

$$\frac{t/u1 \neq \Omega \quad t/u2 = \Omega}{suppr(u, t) = t[(t/u1)/u]} \qquad \frac{t/u1 = \Omega \quad t/u2 \neq \Omega}{suppr(u, t) = t[(t/u2)/u]}$$

$$\frac{t/u1 = \Omega \quad t/u2 = \Omega}{suppr(u, t) = t[\Omega/u]} \qquad \frac{t/u1 \neq \Omega \quad t/u2 \neq \Omega \quad succ(t(u)) = t(v)}{suppr(u, t) = (suppr(v, t))[t(v)/_{\circ}u]}$$

Pour la terminaison, on observe qu'il ne peut y avoir au plus qu'un appel récursif à *suppr*, car l'hypothèse de la quatrième règle implique $succ(t(u)) = min(t/u2)$, puis $t/v1 = \Omega$. Il reste à voir que $suppr(u, t)$ est un arbre binaire de recherche. La substitution du noeud $t(v)$ au noeud u n'introduit pas de violation des conditions, ni à gauche, car $t(u) < t(v)$, ni à droite, car pour tout noeud $w \geq u2$ autre que v on a $t(w) > t(u)$ (par définition du successeur). Les autres cas sont évidents.

CQFD

Arbres rouge et noir. On modifie la syntaxe des termes. Il y a maintenant, pour chaque n, deux opérateurs d'arité 2 : n^N ("noir") et n^R ("rouge"). Il n'y a toujours qu'une constante, qui est noire et est notée Ω^N (ou simplement Ω). Si $t(u) = n^R$ ($t(u) = n^N$), u est appelée occurrence rouge (noire) de t. On définit pour tous u, v tels que $t/v = \Omega$ et $u \leq v$ (ordre préfixe) le nombre $\sharp^N(u, v)$ comme suit :

$$\frac{}{\sharp^N(v, v) = 0} \qquad \frac{ui \leq v \quad t(u) \text{ noir}}{\sharp^N(u, v) = \sharp^N(ui, v) + 1} \qquad \frac{ui \leq v \quad t(u) \text{ rouge}}{\sharp^N(u, v) = \sharp^N(ui, v)}$$

C'est le nombre de noeuds noirs sur le chemin qui mène de u à v (le noeud v non compris). Un arbre rouge et noir est un terme satisfaisant les conditions suivantes :

1. L'arbre sous-jacent obtenu en supprimant les décorations rouge et noir est un arbre binaire de recherche.
2. Si u est rouge, alors $u1$ et $u2$ sont noirs.
3. Pour tout noeuds u, v_1, v_2 tels que $u \leq v_1$, $u \leq v_2$, $t/v_1 = \Omega$ et $t/v_2 = \Omega$, on a $\sharp^N(u, v_1) = \sharp^N(u, v_2)$. La valeur commune est notée $\sharp^N(u)$ (hauteur noire).

(B) Quelle est la complexité des algorithmes en (A) ? Quelle amélioration peut-on espérer avec la discipline rouge et noir ? Justifier en montrant un résultat de la forme $h = O(f(n))$ (hauteur versus taille).

PREUVE. On peut commencer par donner une définition alternative des arbres rouge et noir, par récurrence :

$$\frac{}{\Omega^N \ \text{noir}(0)} \qquad \frac{s \ \text{noir}(k) \quad t \ \text{noir}(k)}{n^R(s,t) \ \text{rouge}(k)}$$

$$\frac{s \ \text{rouge}(k)}{s \ \text{rouge-ou-noir}(k)} \qquad \frac{s \ \text{noir}(k)}{s \ \text{rouge-ou-noir}(k)}$$

$$\frac{s \ \text{rouge-ou-noir}(k) \quad t \ \text{rouge-ou-noir}(k)}{n^N(s,t) \ \text{noir}(k+1)}$$

On désigne par $\mathit{taille}(t)$ le nombre de noeuds de t. C'est aussi la taille de la preuve de (t rouge-ou-noir(k)). On montre que $\mathit{taille}(t) \geq 2^k - 1$:

$$\begin{aligned}
&\mathit{taille}(n^R(s,t)) > \mathit{taille}(s) \geq 2^k - 1 \\
&\mathit{taille}(n^N(s,t)) = \mathit{taille}(s) + \mathit{taille}(t) + 1 \geq (2^k - 1) + (2^k - 1) + 1 = 2^{k+1} - 1 \ .
\end{aligned}$$

Par ailleurs, la seconde règle implique que sur un chemin d'un noeud vers la racine, il y a au moins autant de noeuds noirs que de noeuds rouges. Donc, si h est la profondeur de t (longueur la plus longue d'un chemin depuis la racine), on a $\mathit{taille}(t) \geq (2^k - 1) \geq (2^{h/2} - 1)$ (d'où il résulte que $h = O(\log n)$).
Les algorithmes en (A) sont linéaires en h, donc en n. Si on peut les adapter aux arbres rouge et noir de telle sorte qu'ils restent linéaires en h, ils seront en $O(\log n)$. CQFD

Les deux exercices suivants sont sans incidence sur la suite.

(C1) Peut-on changer la couleur de la racine d'un arbre rouge et noir ?

(C2) Montrer que dans un arbre rouge et noir le rapport de longueur entre deux chemins quelconques de même origine vers deux feuilles est ≤ 2.

Preuve. (C1) Si la racine est rouge, la changer augmente le $\sharp^N$ de 1 sur tous les chemins depuis la racine, donc le nouvel arbre est encore rouge et noir. Si la racine est noire, idem, mais seulement dans le cas où 1 et 2 sont noirs.

(C2) Soit k la hauteur noire commune des deux chemins, et r_1 et r_2 les nombres respectifs de noeuds rouges. On doit montrer $(k + r_1) \leq 2(k + r_2)$, i.e., $r_1 \leq (k + 2r_2)$, qui résulte a fortiori de ce que $r_1 \leq k$. CQFD

(D) Adapter l'algorithme d'insertion au cas rouge et noir (correction et terminaison comprises). Montrer qu'il y a au moins un noeud rouge dans le résultat.

Preuve. On procède d'abord comme en (A), et on donne au noeud u_0 inséré la couleur rouge, de sorte à ne pas modifier la condition (3). Soit t_0 le terme obtenu. Si le père de u_0 est noir, l'arbre obtenu est encore rouge et noir. Si le père v_0 est rouge (ce qui implique que le grand-père w_0 est noir), la condition (2) est violée. Le système formel ci-dessous spécifie comment arriver à un arbre rouge et noir à partir de t_0.

$$(1)\quad \frac{u = vi \quad t(v)\text{ noir}}{(t,u) \;\to !\; t} \qquad\qquad (2)\quad \frac{u = i \quad t = n^R(s_1,s_2)}{(t,u) \;\to !\; n^N(s_1,s_2)}$$

$$(3)\quad \frac{u = wij \quad t/w = k^N(n_1^R(s_1,t_1), n_2^R(s_2,t_2))}{(t,u) \;\to\; (t[k^R(n_1^N(s_1,t_1), n_2^N(s_2,t_2))/w], w)}$$

$$(4)\quad \frac{u = w12 \quad t/w1 = n^R(s_1, m^R(s_2,s_3)) \quad t/w2\text{ noir}}{(t,u) \;\to\; (t[m^R(n^R(s_1,s_2),s_3)/w1], w11)}$$

$$(5)\quad \frac{u = w11 \quad t/w = k^N(n^R(s_1,s_2),s_3) \quad s_3\text{ noir}}{(t,u) \;\to !\; t[n^N(s_1, k^R(s_2,s_3))/w]}$$

$$(6)\quad \frac{u = w21 \quad t/w2 = n^R(m^R(s_1,s_2),s_3) \quad t/w2\text{ noir}}{(t,u) \;\to\; (t[m^R(s_1, n^R(s_2,s_3))/w1], w22)}$$

$$(7)\quad \frac{u = w22 \quad t/w = k^N(s_1, n^R(s_2, s_3)) \quad s_1 \text{ noir}}{(t,u)\ \to !\ t[n^N(k^R(s_1, s_2), s_3)/w]}$$

L'algorithme consiste à réécrire des couples (t, u) formé d'un arbre (la modification courante de l'arbre de départ), et d'un noeud de cet arbre dont la couleur met en défaut la condition (2). (Nous conseillons au lecteur de faire les dessins correspondant aux règles.)

Les conditions (1) et (3) sont préservées à chaque réécriture (facile à voir sur les motifs présents dans les règles). On vérifie ensuite que si $(t_0, u_0) \to^*!\ t$, alors t vérifie la condition (2). Nous examinons le cas de la règle (5). On a supprimé la situation rouge-rouge en $w1$ et $w11$, mais on a introduit le rouge en $w2$: il faut donc s'assurer que s_2 est noir (s_3 l'est par hypothèse). Or $w1$ satisfaisait la condition (2) dans l'arbre de départ (l'algorithme ne fait que remonter dans l'arbre, ce qui est "au-dessus" est donc toujours rouge et noir), donc s_2, qui était au départ fils de noeud rouge, était noir et l'est resté.

Pour la terminaison, on utilise le fait que le seul "vrai" appel récursif fait remonter dans l'arbre (règle (3), passage de wij à w). Les autres appels récursifs (4) et (6) sont juste des ajustements immédiatement suivis des règles terminales (5) et (7). Enfin, les règles partent d'un t_0 possédant au moins un noeud rouge (celui qui vient d'être inséré). Toutes les règles maintiennent l'existence d'au moins un noeud rouge : le u' tel que $(t, u) \;\to\; (t', u')$ pour les règles non terminales, u lui-même pour les règles (1) et (2), et $w2$ et $w1$ respectivement pour les règles (5) et (7). CQFD

14 Bases de Gröbner

Source. "Term rewriting and all that" de Franz Baader et Tobias Nipkow (Cambridge University Press 1998).

On se place dans $K[X_1, \ldots, X_n]$, anneau de polynômes à plusieurs variables (K étant le corps des réels, ou n'importe quel corps commutatif). Rappelons qu'un polynôme f s'écrit comme une somme $\sum a_i m_i$ (avec les m_i distincts) où chaque m_i est un monôme $X_1^{k_1} \cdots X_n^{k_n}$ (avec les $k_j \geq 0$). Le degré d'un tel monôme est $\sum k_j$. On dit que a_i est le coefficient de m_i dans f. On note (idéal engendré) :

$$\langle f_1, \ldots, f_k \rangle = \{f_1 g_1 + \cdots + f_k g_k \mid g_1, \ldots, g_k \in K[X_1, \ldots, X_n]\} .$$

Si J est un idéal (i.e., un sous-ensemble clos par addition et par multiplication par un polynôme quelconque), on note $f \equiv_J g$ si et seulement si $f - g \in J$.
Le problème est : étant donné $J = \langle f_1, \ldots, f_k \rangle$, décider $\equiv_J$, i.e., trouver un algorithme qui prend deux polynômes f et g en entrée et renvoie OUI si $f \equiv_J g$ et NON si $f \not\equiv_J g$. La technique employée est la réécriture. Le principe est le suivant. Si f est un polynôme dans lequel apparaît un monôme m_1 avec le coefficient a, on peut "couper" f en am_1 et $\hat{f} = f - am_1$. On a $am_1 \equiv -\hat{f}$, que l'on regarde comme une règle de réécriture (ou de calcul) $am_1 \to -\hat{f}$. On applique cette règle de la manière suivante : si $g = bm_1m_2 + g_1$ est un polynôme dont l'un des monômes est divisé par m_1, on écrit

$$g \to g' = -(b/a)\hat{f}m_2 + g_1 .$$

On a $g \equiv_J g'$ (car $g' = g - (b/a)fm_2$), avec g' "plus simple" pour peu que l'on ait choisi m_1 "maximal" dans f. On dit que m_1m_2 est le pivot de la réduction. Il faut donc définir un ordre sur les monômes. En voici un ($>_{lex}$ est l'ordre lexicographique) :

$$X_1^{k_1} \cdots X_n^{k_n} \succ X_1^{l_1} \cdots X_n^{l_n} \Leftrightarrow \begin{cases} (\sum k_j > \sum l_j) \text{ ou} \\ (\sum k_j = \sum l_j) \text{ et } ((k_1, \ldots, k_n) >_{lex} (l_1, \ldots, l_n)) . \end{cases}$$

(A) Montrer que $\succ$ est bien fondé (pas de suite infinie strictement décroissante).

PREUVE. On traite le cas $n = 2$ (pour ne pas alourdir les notations). Rappelons que $(k_1, k_2) >_{lex} (l_1, l_2)$ veut dire $k_1 > l_1$ ou ($k_1 = l_1$ et $k_2 > l_2$). Si $(k_1^1, k_2^1), \ldots, (k_1^n, k_2^n), \ldots$ est une suite infinie strictement décroissante pour $\succ$, alors la suite des $k_1^i + k_2^i$ devient forcément stationnaire, donc à partir d'un certain rang les $\succ$ sont en fait des $>_{lex}$. Or l'ordre lexicographique est bien fondé : les k_1^i forment une suite infinie décroissante et par conséquent stationnaire, donc à partir d'un certain rang la composante de gauche reste constante, tandis que la composante de droite décroit strictement : or les nombres entiers naturels sont bien fondés. CQFD

Soit f un polynôme. On note $H(f)$ son plus grand monôme (selon $\succ$). On supposera toujours, sans perte de généralité, que le coefficient de $H(f)$ dans f est 1 (soit $a = 1$ ci-dessus). On pose $R(f) = f - H(f)$. On note $\rightarrow_f$ la relation $\rightarrow$ de ci-dessus obtenue avec ce découpage de f, soit

$$g + bH(f)m \ \rightarrow_f \ g - bR(f)m$$

et, si $G = \{f_1, \ldots, f_k\}$, on note $\rightarrow_G = \bigcup \rightarrow_{f_i}$. Soulignons que $\rightarrow_G$ dépend du choix des générateurs $f_1, \ldots, f_k$ de $J = \langle f_1, \ldots, f_k \rangle$ (cf. la distinction entre un espace vectoriel et une base particulière de cet espace).

(B) Soit $G = \{f_1, \ldots, f_k\}$ et $J = \langle f_1, \ldots, f_k \rangle$. Montrer que $\equiv_J$ est la clôture réflexive, symétrique et transitive $\leftrightarrow_G^*$ de $\rightarrow_G$.

PREUVE. On a par construction $\rightarrow_G \subseteq \equiv_J$, donc $\leftrightarrow_G^* \subseteq \equiv_J$. Réciproquement, si $g \equiv_J h$, on peut passer de g à h par étapes successives d'addition d'un polynôme de la forme kmf, où $k \in K$ et $f \in \{f_1, \ldots, f_k\}$. On est donc ramené à prouver que $(h = g + kmf \Rightarrow h \ \leftrightarrow_G^* \ g)$, ce qui résulte des deux propriétés suivantes : $(kmf \ \rightarrow_f \ 0)$ et

$$\textbf{(P)} \quad (g_1 \ \rightarrow_f \ g_2 \Rightarrow (\exists\, h \ \ g + g_1 \ \rightarrow_f \ h \text{ et } g + g_2 \ \rightarrow_f \ h))\ .$$

La première est immédiate par définition de $\rightarrow_f$. Pour (P), soit $m_1 = H(f)$, soit $a \neq 0, m_2$ tels que $g_2 = g_1 - am_2f$, et soit b le coefficient de m_1m_2 dans g. Alors les coefficients de m_1m_2 dans $g + g_1$ et $g + g_2$ sont respectivement $a + b$ et b. On a donc

$$g + g_1 \ \rightarrow_f^{\leq 1} \ h_1 = g + g_1 - (a + b)m_2f \quad \text{et} \quad g + g_2 \ \rightarrow_f^{\leq 1} \ h_2 = g + g_2 - bm_2f\ ,$$

où $^{\leq 1}$ veut dire une ou zéro étape. En effet, si $a + b \neq 0$, alors il y a bien une étape de réécriture, tandis que si $a + b = 0$ alors h_1 coïncide avec $g + g_1$; et de même pour $g + g_2$. On conclut en observant que

$$h_1 = g + g_1 - (a + b)m_2f = g + g_2 - bm_2f = h_2\ .$$

CQFD

(C) Montrer que la relation $\rightarrow_G$ est bien fondée.

PREUVE. Si $g \rightarrow_G h$, alors l'ensemble $M(h)$ des monômes de h est plus petit que $M(g)$ pour l'ordre multi-ensemble $<_{mul}$ défini par :

$$\frac{m \in X \quad m \succ m_1, \dots, m_k}{X >_{mul} X \cup \{m_1, \dots, m_k\} \setminus \{m\}}$$

(i.e., on remplace un monôme par un ensemble fini de monômes plus petits). Or l'ordre multi-ensemble défini à partir d'un ordre bien fondé est bien fondé. Pour le voir, on construit à partir d'une suite infinie $X_1 >_{mul} \cdots >_{mul} X_n >_{mul} \cdots$ un arbre qui en témoigne, e.g. si $X_{n+1} = X_n \cup \{m_1, \dots, m_k\} \setminus \{m\}$, alors $a_1, \dots, a_k$ sont les fils de a. Remarque, si k=0, il faut prévoir un noeud "bouchon", histoire d'être sûr que l'arbre croît toujours. On obtient donc un arbre infini à branchement fini. Un tel arbre a au moins une branche infinie (cette propriété, appelée lemme de König, est l'une des formes de l'axiome du choix). Or, par construction et par définition de $<_{mul}$, on décroît strictement en descendant le long d'une branche de l'arbre, ce qui contredit (A). CQFD

On pose ensuite la définition suivante : $G = \{f_1, \dots, f_k\}$ est appelé base de Gröbner (pour $J = \langle f_1, \dots f_k \rangle$) si $\rightarrow_G$ est confluent, i.e. si

$$\forall f, g, h \ \ ((f \rightarrow_G^* g \text{ et } f \rightarrow_G^* h) \Rightarrow (\exists k \ \ g \rightarrow_G^* k \text{ et } h \rightarrow_G^* k)) \ .$$

Les deux résultats qui suivent sont des lemmes généraux.

(D1) Montrer que si $\rightarrow_G$ est confluent, alors pour tous f et g, si $f \leftrightarrow_G^* g$, alors f et g ont un réduit commun.

(D2) Montrer que la bonne fondation de $\rightarrow_G$ implique que $\rightarrow_G$ est confluente si et seulement si la propriété suivante (appelée confluence locale) est vérifiée :

$$\forall f, g, h \ \ ((f \rightarrow_G g \text{ et } f \rightarrow_G h) \Rightarrow (\exists k \ \ g \rightarrow_G^* k \text{ et } h \rightarrow_G^* k))$$

(un seul pas de réduction dans chacune des hypothèses).

PREUVE. (D1) On procède par récurrence sur la longueur d'un "zigzag" $f \leftrightarrow_G f_1 \cdots \leftrightarrow_G \cdots \leftrightarrow_G g$, où, disons $f \leftrightarrow_G f_1$ veut dire $f \rightarrow_G f_1$ ou $f_1 \rightarrow_G f$. Par récurrence, f_1 et g ont un réduit commun h. Si $f \rightarrow_G f_1$, alors h est un réduit commun de f et g. Si $f_1 \rightarrow_G f$, alors par confluence f et h ont un réduit commun, qui est aussi un réduit commun de f et g.

(D2) On procède par récurrence sur la longueur maximale d'une réduction $f \rightarrow_G^* k$ à partir de f, que l'on note $\mathit{prof}(f)$. La propriété (C) dit que $\mathit{prof}(f)$ est toujours fini. Si $f \rightarrow_G g_1 \rightarrow_G^* g$ et $f \rightarrow_G h_1 \rightarrow_G^* h$, alors $\mathit{prof}(g_1) < \mathit{prof}(f)$ et $\mathit{prof}(h_1) < \mathit{prof}(f)$. Par ailleurs, par confluence locale, g_1 et h_1 ont un réduit commun k_1, qui vérifie aussi $\mathit{prof}(k_1) < \mathit{prof}(f)$. Puis, par récurrence appliquée à g_1 (h_1), g et k_1 (h et k_1) ont un réduit commun g' (h'). Enfin, par récurrence appliquée à k_1, g' et h' ont un réduit commun, et a fortiori g et h aussi. CQFD

On retourne aux polynômes. La question suivante montre l'intérêt des bases de Gröbner.

(E1) Montrer que si $G = \{f_1, \ldots, f_n\}$ est une base de Gröbner pour $J = \langle f_1, \ldots, f_k \rangle$, alors $\equiv_J$ est décidable.

(E2) Trouver des polynômes f_1, f_2 tels que $\{f_1, f_2\}$ n'est pas une base de Gröbner.

(E3) On pose (pour tous g_1, g_2) $S(g_1, g_2) = m_1 g_1 - m_2 g_2$, où m_1 et m_2 sont tels que $m_1 H(g_1) = m_2 H(g_2)$ est le plus petit commun multiple de $H(g_1)$ et $H(g_2)$. Montrer que $G = \{f_1, \ldots, f_k\}$ est une base de Gröbner si et seulement si $(S(f_i, f_j) \rightarrow_G^\star 0)$, pour toutes les paires i, j.

Preuve. (E1) Pour décider $g_1 \equiv_J g_2$, il suffit de calculer une forme normale h_1 (h_2) de g_1 (g_2), i.e. $g_1 \rightarrow_G^* h_1$ et $\not\exists k \; h_1 \rightarrow_G k$. Alors $g_1 \equiv_J g_2$ si et seulement si h_1 et h_2 coïncident. En effet, puisque $h_1 \equiv_J h_2$, ils doivent avoir un réduit commun par (D1), qui ne peut que coïncider respectivement avec h_1 et h_2 puisque h_1 et h_2 sont en forme normale.

(E2) Prendre $f_1 = X_1^2 X_2 - X_1^2$ et $f_2 = X_1 X_2^2 - X_2^2$: on a $X_1^2 X_2^2 \rightarrow^* X_1^2$ et $X_1^2 X_2^2 \rightarrow^* X_2^2$, qui sont deux formes normales.

(E3)($\Rightarrow$) Par définition, on a $S(f_i, f_j) \equiv_J 0$. Donc, par (B), $S(f_i, f_j) \leftrightarrow_G^\star 0$, et par (D1), on en déduit que $S(f_i, f_j)$ et 0 ont un réduit commun. Comme 0 est une forme normale, on a bien $S(f_i, f_j) \rightarrow_G^\star 0$.

(E3)($\Leftarrow$) Par (D2), il suffit d'examiner les situations $f \rightarrow_{f_i} g$ et $f \rightarrow_{f_j} h$. Soient $H(f_i)m_i$ et $H(f_j)m_j$ les pivots respectifs de ces deux réductions, de coefficients respectifs a et b dans f.

1. Si les deux pivots sont différents, on a $f = f_1 + aH(f_i)m_i + bH(f_j)m_j$, $g = f - am_i f_i$ et $h = f - bm_j f_j$. On peut supposer $H(f_i)m_i \succ H(f_j)m_j$, le problème étant symétrique. Ceci garantit que $H(f_i)m_i$ apparaît dans h avec le coefficient a, car la réécriture $f \rightarrow h$ n'introduit que des monômes $\prec H(f_j)m_j$. On a donc $h \rightarrow_{f_i} h - am_i f_i$. On conclut alors ce cas en utilisant

la propriété (P) (cf. preuve de (B)) : $g = f - am_i f_i$ et $h - am_i f_i$ ont un réduit commun, et donc a fortiori g et h aussi.

2. Si le pivot est le même pour les deux réductions, on pose $H(f_i)m_i = H(f_j)m_j$. On a $a = b$, $g = f - am_i f_i$ et $h = f - am_j f_j$. Soient m_1 et m_2 tels que $m_1 H(f_i) = m_2 H(f_j)$ est le plus petit commun multiple de $H(f_i)$ et $H(f_j)$. Alors il existe m' tel que $m_i = m'm_1$ et $m_j = m'm_2$, et $g - h = am'(m_2 f_j - m_1 f_i) = am' S(f_j, f_i)$. Il ne reste plus qu'à montrer la propriété suivante :

$$(\mathbf{Q}) \quad (g - h \rightarrow^{\star}_{G} 0) \Rightarrow (g \text{ et } h \text{ ont un réduit commun}) .$$

La preuve est par récurrence sur la longueur de la réécriture de $g - h$ vers 0. Si le pivot de la première étape est $H(f_i)m'$, et si a, b sont les coefficients de $H(f_i)m'$ dans g et h, alors $g - h \rightarrow_{f_i} (g - am'f_i) - (h - bm'f_i)$; par récurrence $g - am'f_i$ et $h - bm'f_i$ ont un réduit commun, et on conclut en remarquant que $g \rightarrow^{\leq 1}_{f_i} g - am'f_i$ (cf. preuve de (P)). CQFD

L'étape finale du problème consiste à trouver une base de Gröbner pour un idéal donné $J = \langle f_1, \ldots, f_k \rangle$. Si $\{f_1, \ldots, f_k\}$ satisfait déjà le critère (E3), il n'y a rien à faire. Sinon, on utilise l'algorithme suivant. On pose $(G_0, B_0) = (\{f_1, \ldots, f_k\}, \{(f_l, f_r) \mid 1 \leq l < r \leq k\})$ et on définit le système formel suivant ($\underline{h}$ désigne le polynôme kh ($k \in K$) tel que le coefficient du monôme de tête ait le coefficient 1) :

$$\frac{(f_l, f_r) \in B \quad S(f_l, f_r) \rightarrow^{\star}_{G} h \neq 0 \text{ et irréductible pour } \rightarrow_G}{(G, B) \longrightarrow (G \cup \{\underline{h}\}, B \cup \{(\underline{h}, f) \mid f \in G\} \setminus \{(f_l, f_r)\})}$$

$$\frac{(f_l, f_r) \in B \quad S(f_l, f_r) \rightarrow^{\star}_{G} 0}{(G, B) \longrightarrow (G, B \setminus \{(f_l, f_r)\})}$$

(F1) Montrer que si $(G, B) = (G_p, B_p)$ est une forme normale de (G_0, B_0) pour ce système, alors G est une base de Gröbner pour J.

(F2) Montrer que le système termine toujours, en admettant le résultat suivant (objet de la question (G)) : si $m_1, \ldots, m_n, \ldots$ est une suite infinie de monômes, alors il existe $i < j$ tel que m_i divise m_j.

PREUVE. (F1) A chaque étape $(G_n, B_n) \longrightarrow (G_{n+1}, B_{n+1})$, $G_{n+1} = G_n$ ou $G_{n+1} = G_n \cup \{\underline{h}\}$, où $\underline{h}$ est une combinaison de deux membres de G_n. Donc G_n

engendre le même idéal que G_{n+1}, et, de proche en proche, G engendre J que G_0 engendre. Il reste à montrer $(\forall f_l, f_r \in G \;\; S(f_l, f_r) \;\to_G^\star\; 0)$ (cf. (E3)). Puisque (G, B) est une forme normale, $B = \emptyset$, sinon l'une des deux règles s'appliquerait. On utilise l'invariant suivant : pour tout n, pour tous $g_1, g_2 \in G_n$ distincts, le couple (g_1, g_2) ou le couple (g_2, g_1) est apparu dans un B_m antérieur $(m \leq n)$: c'est vrai par construction en (G_0, B_0), et la conclusion de la première règle réinsère tous les nouveaux couples à considérer. Supposons qu'il existe $f_l, f_r \in G$ tels que $S(f_l, f_r)$ a une forme normale $h \neq 0$ pour $\to_G$. Par l'invariant, $(f_l, f_r) \in B_m$ pour au moins un m, que l'on prend maximal. On a $m < p$ puisque $B_p = B = \emptyset$. Par maximalité de m, l'étape $(G_m, B_m) \longrightarrow (G_{m+1}, B_{m+1})$ enlève (f_l, g_r). Puisque a fortiori la forme normale h' de $S(f_l, f_r)$ pour $\to_{G_m}$ ne peut être 0, cette $(m+1)$-ème étape emprunte la première règle. Donc $h' \in G_{m+1}$, ce qui implique que la forme normale de $S(f_l, f_r)$ pour G_{m+1} (et a fortiori pour G : contradiction !) est 0 :

$$S(f_l, f_r) \;\to_{G_m}^\star\; h' \;\to_{G_{m+1}}\; 0 \,.$$

(F2) Une dérivation infinie $(G_0, B_0) \longrightarrow \cdots \longrightarrow (G_n, B_n) \longrightarrow \cdots$ doit emprunter une infinité de fois la première règle : en effet, si à partir d'un certain (G_m, B_m) on n'applique plus que la seconde règle, on ne touche plus à G_m et on diminue strictement le cardinal de la seconde composante à chaque fois. On construit donc une suite infinie $\underline{h_{i_1}}, \ldots, \underline{h_{i_n}}, \ldots$ et une suite infinie d'ensembles $G_{i_1}, \ldots, G_{i_n}, \ldots$ tels que $G_{i_{n+1}} = G_{i_n} \cup \{\underline{h_{i_n}}\}$ et h_{i_n} est en forme normale relativement à G_{i_n}. En particulier, comme $\underline{h_{i_m}} \in G_{i_n}$ pour tout $m < n$, on ne peut avoir que $H(\underline{h_{i_m}})$ divise un monôme de $\underline{h_{i_n}}$, et parmi eux $H(\underline{h_{i_n}})$. CQFD

Pour prouver la propriété utilisée en (F2), il reste un petit peu de travail, et cela passe par la notion de bon ordre partiel. Un bon ordre partiel est un ensemble ordonné X tel que pour toute suite infinie $x_1, \ldots, x_n, \ldots$ d'éléments de X, il existe i, j tels que $i < j$ et $x_i \leq x_j$. Cette propriété implique la bonne fondation, car s'il existait une suite $x_1, \ldots, x_n, \ldots$ telle que $x_n > x_{n+1}$ pour tout n, alors on ne pourrait avoir $i < j$ tels que $x_i \leq x_j$ (antisymétrie de $\leq$). Réciproquement, si l'ordre est total, il est facile de voir que la bonne fondation implique la propriété de bon ordre partiel.

Si X et Y sont des ordres partiels, l'ordre produit sur $X \times Y$ est défini par $(x_1, y_1) \leq (x_2, y_2)$ si et seulement si $x_1 \leq x_2$ et $y_1 \leq y_2$. Il est facile de voir que si X et Y sont bien fondés, alors $X \times Y$ est bien fondé. Mais c'est le résultat plus fort que $X \times Y$ est un bon ordre partiel qui nous intéresse (noter que si X et Y sont des ordres totaux, $X \times Y$ n'est pas total sauf cas trivial : $(3, 5)$ versus $(4, 2)$).

(G1) Montrer que dans un bon ordre partiel X tout suite infinie $x_1, \ldots, x_n, \ldots$ admet une sous-suite croissante $x_{i_1} \leq \cdots \leq x_{i_n} \leq \cdots$.

(G2) Prouver que si X et Y sont des bons ordres partiels, alors $X \times Y$ est un bon ordre partiel.

(G3) Prouver l'hypothèse faite dans la question (F2).

PREUVE. (G1) On fixe une suite $x_1, \ldots, x_n, \ldots$. On dit que m est terminal dans la suite si $(\nexists n > m\ x_m \leq x_n)$. Supposons qu'il y ait une infinité $m_1 < \cdots < m_n < \cdots$ d'indices terminaux. Considérons la suite $x_{m_1}, \ldots, x_{m_n}, \ldots$. Par hypothèse de bon ordre partiel appliquée à cette suite, il existe i, j tels que $i < j$ et $x_{m_i} \leq x_{m_j}$. Mais ceci contredit le fait que x_{m_i} est terminal. Donc, en enlevant suffisamment d'éléments initiaux de la suite $x_1, \ldots, x_n, \ldots$, on peut supposer qu'aucun indice n'est terminal. On définit alors une sous suite croissante $x_{i_1} \leq \cdots \leq x_{i_n} \leq \cdots$ comme suit : $x_{i_1} = x_1$, et si x_{i_n} a été défini, on choisit $x_{i_{n+1}}$ tel que $x_{i_n} \leq x_{i_{n+1}}$, ce qui est toujours possible puisque aucun élément de la suite n'est terminal.

(G2) Par (G1) appliqué à X, il est suffisant de considérer une suite infinie $(x_1, y_1), \ldots, (x_n, y_n) \ldots$ dont la projection sur X est croissante, i.e., $x_1 \leq \cdots \leq x_n \leq \cdots$. Comme Y est un bon ordre partiel, il existe i, j tels que $i < j$ et $y_i \leq y_j$, et donc $(x_i, y_i) \leq (x_j, y_j)$.

(G3) Le résultat (G2) sétend à un produit de n bons ordres partiels. La relation de divisibilité entre monômes est à isomorphisme près l'ordre produit de l'ordre sur les entiers naturels. CQFD

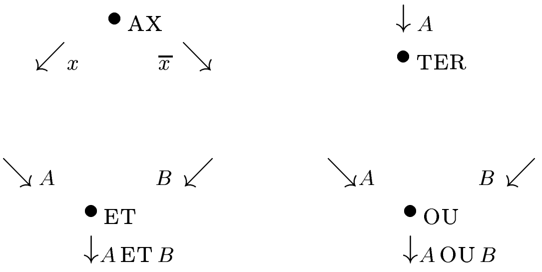
15 Réseaux multiplicatifs de Girard

Source. J.-Y. Girard, “Linear logic”, Theoretical Computer Science 50 (1987).

On construit des graphes orientés dont les noeuds et les arêtes sont étiquetés. Formellement, on appelle préréseau un graphe orienté $G = (S, A)$ avec deux fonctions d'étiquetage $\epsilon_S : S \to \{$ AX, OU , ET , TER$\}$ et $\epsilon_A : A \to T$, où T est l'ensemble des formules construites à partir de littéraux x ou $\overline{x}$, de ET et de OU . Les contraintes suivantes doivent être satisfaites (on note $\epsilon_S = \epsilon_A = \epsilon$, et, plus loin, on écrira souvent les étiquettes au-dessus des noeuds ou arêtes).

- Si $\epsilon(u) =$ AX, alors u n'a pas d'arêtes entrantes, et a exactement deux arêtes sortantes (u, v_1) et (u, v_2), et $\epsilon(u, v_1)$ et $\epsilon(u, v_2)$ sont deux littéraux opposés.
- Si $\epsilon(u) =$ ET , alors u a exactement deux arêtes entrantes (v_1, u) et (v_2, u) et une arête sortante (u, v_3), et $\epsilon(u, v_3) = (\epsilon(v_1, u)\,\text{ET}\,\epsilon(v_2, u))$.
- Si $\epsilon(u) =$ OU , alors u a exactement deux arêtes entrantes (v_1, u) et (v_2, u) et une arête sortante (u, v_3), et $\epsilon(u, v_3) = (\epsilon(v_1, u)\,\text{OU}\,\epsilon(v_2, u))$.
- Si $\epsilon(u) =$ TER, alors u a exactement une arête entrante (v, u).

Graphiquement :



(A) Montrer (informellement) qu'un graphe ainsi construit peut être vu comme un ensemble $\{A_1, \ldots, A_n\}$ de formules de T plus la donnée d'une partition de l'ensemble des occurrences de littéraux dans $A_1, \ldots, A_n$ en classes composées chacune de deux littéraux opposés.

PREUVE. En disposant les noeuds et les arêtes en suivant les contraintes, on place les AX en haut, Puis viennent les noeux ET et OU , et enfin les noeuds TER. A partir de chaque noeud TER on développe un arbre (dont la tête est en bas et les feuilles aboutissent aux noeuds AX. L'appariement fourni par les noeuds AX correspond alors à la condition de l'énoncé sur les occurrences de littéraux.

CQFD

On appelle réseau séquentiel tout préréseau construit exclusivement à partir des règles qui suivent (exprimées à l'aide de la formulation des préréseaux donnée en (A)) :

(R1) Pour tout littéral x, $\{x, \overline{x}\}$ est un réseau séquentiel, et la partition n'a qu'une seule classe.

(R2) Si $\{A_1, \ldots, A_n, A\}$ et $\{B, B_1, \ldots, B_m\}$ sont des réseaux séquentiels, alors $\{A_1, \ldots, A_n, (A \,\textsc{et}\, B), B_1, \ldots, B_m\}$ est un réseau séquentiel, et la partition est formée de l'union des classes.

(R3) Si $\{A_1, \ldots, A_n, A, B\}$ est un réseau séquentiel, alors $\{A_1, \ldots, A_n, (A \,\textsc{ou}\, B)\}$ est un réseau séquentiel, et la partition est inchangée.

(B1) Traduire la définition ci-dessus en termes de dessins, puis formellement en termes de graphes orientés. (Un point de vocabulaire : on appelle conclusion de (S, A) toute arête (ou son étiquette) de A vers un noeud TER de S.)

(B2) Parmi les préréseaux suivants, lesquels sont des réseaux séquentiels ? Y a-t-il plusieurs manières de l'établir (c'est ce qu'on appelle séquentialiser un réseau) ?

1. $\{x \,\textsc{ou}\, \overline{x}\}$
2. $\{x \,\textsc{et}\, \overline{x}\}$
3 $\{x, \overline{x}, y, \overline{y}\}$
4. $\{(x \,\textsc{et}\, y), (\overline{x} \,\textsc{ou}\, \overline{y})\}$
5. $\{(x \,\textsc{et}\, y), (\overline{x} \,\textsc{et}\, \overline{y})\}$
6. $\{(x \,\textsc{ou}\, y), (\overline{x} \,\textsc{ou}\, \overline{y})\}$
7. $\{x, (\overline{x} \,\textsc{et}\, \overline{y}), (y \,\textsc{et}\, (z \,\textsc{ou}\, u)), (\overline{z} \,\textsc{et}\, \overline{u})\}$

(B3) Expliciter le pourquoi des noms ET et OU, i.e., exhiber un lien entre les règles de construction des réseaux séquentiels et la signification habituelle de la conjonction et de la disjonction.

Preuve. (B1) Règle (R1) : tout graphe

$$(\{\overset{\text{AX}}{u}, \overset{\text{TER}}{v_1}, \overset{\text{TER}}{v_2}\}, \{\overset{x}{(u, v_1)}, \overset{\overline{x}}{(u, v_2)}\})$$

est un réseau séquentiel. Règle (R2) : si (S_1, A_1) et (S_2, A_2) sont des réseaux séquentiels disjoints et si (u_1, v_1) et (u_2, v_2) sont des arêtes conclusion de (S_1, A_1) et (S_2, A_2) d'étiquettes A et B respectivement, alors le graphe suivant (S, A) est séquentiel (u_3, v_3 sont de nouveaux sommets) :

$$S = S_1 \cup S_2 \cup \{\overset{\text{ET}}{u_3}, \overset{\text{TER}}{v_3}\} \setminus \{\overset{\text{TER}}{v_1}, \overset{\text{TER}}{v_2}\}$$

$$A = A_1 \cup A_2 \cup \{\overset{A}{(u_1, u_3)}, \overset{B}{(u_2, u_3)}, \overset{A\,\text{ET}\,B}{(u_3, v_3)}\} \setminus \{\overset{A}{(u_1, v_1)}, \overset{B}{(u_2, v_2)}\}$$

Règle (R3) : si (S_1, A_1) est un réseau séquentiel et si (u_1, v_1) et (u_2, v_2) sont deux arêtes conclusion distinctes de (S_1, A_1) d'étiquettes A et B respectivement, alors le graphe suivant (S, A) est séquentiel (u_3, v_3 sont de nouveaux sommets) :

$$S = S_1 \cup \{\overset{\text{OU}}{u_3}, \overset{\text{TER}}{v_3}\} \setminus \{\overset{\text{TER}}{v_1}, \overset{\text{TER}}{v_2}\}$$

$$A = A_1 \cup \{\overset{A}{(u_1, u_3)}, \overset{B}{(u_2, u_3)}, \overset{A\,\text{OU}\,B}{(u_3, v_3)}\} \setminus \{\overset{A}{(u_1, v_1)}, \overset{B}{(u_2, v_2)}\}$$

(B2) Dans la liste, les réseaux séquentiels sont les suivants :

- Préréseau 1 : appliquer (R3) à $\{x, \overline{x}\}$.
- Préréseau 4. On obtient $\{(x\,\text{ET}\,y), y, \overline{y}\}$ en appliquant (R2) à $\{x, \overline{x}\}$ et $\{y, \overline{y}\}$, puis on applique (R3).
- Préréseau 7 : Il faut former $(\overline{z}\,\text{ET}\,\overline{u})$ avant $(z\,\text{OU}\,u)$, et former ce dernier avant $(y\,\text{ET}\,(z\,\text{OU}\,u))$. On peut former $(\overline{x}\,\text{ET}\,\overline{y})$ quand on veut. En particulier, on peut former $(\overline{x}\,\text{ET}\,\overline{y})$ ou $(y\,\text{ET}\,(z\,\text{OU}\,u))$ en dernier.

Justification informelle du rejet des autres réseaux :

- Préréseau 2 : il faudrait deux graphes disjoints pour construire le noeud ET .
- Préréseau 3 : on n'a pas donné de règle pour faire l'union de deux graphes tout court.
- Préréseaux 5 et 6 : Noter l'ordre de la justification pour le préréseau 4 : d'abord relier deux réseaux (avec ET), et ensuite construire le OU (il n'y a plus qu'un seul réseau).

(B3) On interprète $\{A_1, \ldots, A_n\}$ comme la disjonction des A_i. On montre, par récurrence sur la formation du réseau, que cette disjonction est une tautologie (i.e., est toujours vérifiée) en tant que formule de la logique propositionnelle. La règle (R1) est le tiers exclu (x ou sa négation est toujours vrai). La règle (R3) est un ajustement associatif de parenthèses (si A', A ou B sont vrais, alors A' ou $(A\,\text{OU}\,B)$ est vrai). Enfin, pour la règle (R2), un petit jeu de tables de vérité emporte la conviction : on peut prendre $n = m = 1$; si $(A_1\,\text{OU}\,A)$ est vrai, et si $(B_1\,\text{OU}\,B)$ est vrai, alors $(A_1\,\text{OU}\,A)\,\text{ET}\,(B_1\,\text{OU}\,B) = A_1\,\text{OU}\,B_1\,\text{OU}\,(A\,\text{ET}\,B)$. CQFD

Il existe une autre caractérisation des réseaux séquentiels, plus globale et plus géométrique. Pour l'exprimer, on introduit deux notions : les interrupteurs et les voyages. Soit $G = (S, A)$ un préréseau. On appose arbitrairement sur tous les noeuds ET et OU une étiquette supplémentaire G ou D. Un noeud ainsi annoté est considéré comme une position d'interrupteur déterminant de manière unique un cheminement dans le graphe, selon les règles suivantes (dans lesquelles $(u,v) \to (v,w)$ se lit "après (u,v), emprunter (v,w)") :

$$\frac{\overset{A}{(v,u)} \quad \overset{\mathrm{ET}_G}{u} \quad \overset{A\,\mathrm{ET}\,B}{(u,w)}}{(v,u) \to (u,w)} \qquad \frac{\overset{A\,\mathrm{ET}\,B}{(u,w)} \quad \overset{\mathrm{ET}_G}{u} \quad \overset{B}{(v,u)}}{(w,u) \to (u,v)} \qquad \frac{\overset{B}{(v_2,u)} \quad \overset{\mathrm{ET}_G}{u} \quad \overset{A}{(v_1,u)}}{(v_2,u) \to (u,v_1)}$$

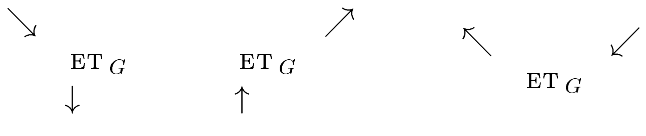
$$\frac{\overset{B}{(v,u)} \quad \overset{\mathrm{ET}_D}{u} \quad \overset{A\,\mathrm{ET}\,B}{(u,w)}}{(v,u) \to (u,w)} \qquad \frac{\overset{A\,\mathrm{ET}\,B}{(u,w)} \quad \overset{\mathrm{ET}_D}{u} \quad \overset{A}{(v,u)}}{(w,u) \to (u,v)} \qquad \frac{\overset{A}{(v_1,u)} \quad \overset{\mathrm{ET}_D}{u} \quad \overset{B}{(v_2,u)}}{(v_1,u) \to (u,v_2)}$$

$$\frac{\overset{A}{(v,u)} \quad \overset{\mathrm{OU}_G}{u} \quad \overset{A\,\mathrm{OU}\,B}{(u,w)}}{(v,u) \to (u,w)} \qquad \frac{\overset{A\,\mathrm{OU}\,B}{(u,w)} \quad \overset{\mathrm{OU}_G}{u} \quad \overset{A}{(v,u)}}{(w,u) \to (u,v)} \qquad \frac{\overset{B}{(v,u)} \quad \overset{\mathrm{OU}_G}{u}}{(v,u) \to (u,v)}$$

$$\frac{\overset{B}{(v,u)} \quad \overset{\mathrm{OU}_D}{u} \quad \overset{A\,\mathrm{OU}\,B}{(u,w)}}{(v,u) \to (u,w)} \qquad \frac{\overset{A\,\mathrm{OU}\,B}{(u,w)} \quad \overset{\mathrm{OU}_D}{u} \quad \overset{B}{(v,u)}}{(w,u) \to (u,v)} \qquad \frac{\overset{A}{(v,u)} \quad \overset{\mathrm{OU}_D}{u}}{(v,u) \to (u,v)}$$

$$\frac{\overset{x}{(u,v_1)} \quad \overset{\mathrm{AX}}{u} \quad \overset{\overline{x}}{(u,v_2)}}{(v_1,u) \to (u,v_2)} \qquad \frac{\overset{\mathrm{TER}}{v}}{(u,v) \to (v,u)}$$

Graphiquement, par exemple pour le ET gauche :



Le lecteur est invité à faire un dessin pour les autres cas ! On peut noter que le parcours est bien déterministe et ne s'arrête jamais. On dit qu'un préréseau est un G-réseau si et seulement si, pour tout choix d'interrupteurs, une fois ce choix fixé, et une fois choisie une arête arbitraire (u,v), la réécriture partant de

(u,v) revient pour la première fois à (u,v) en ayant parcouru toutes les arêtes du graphe une et une seule fois dans chaque sens. C'est ce qu'on appelle un long voyage.

(C1) Montrer que tout réseau séquentiel est un G-réseau.

(C2) Montrer que les réseaux rejetés en (B2) ne sont pas des G-réseaux.

(C3) Montrer qu'un G-réseau est un réseau séquentiel (en identifiant et en laissant de côté le cas difficile).

PREUVE. (C1) Par récurrence sur la construction des réseaux séquentiels. Pour (R1), on a :

$$(u,v_1) \to (v_1,u) \to (u,v_2) \to (v_2,u) \to (u,v_1)\ .$$

Pour (R2), avec les notations de la preuve de (B1), on se place sur l'arête (u_3,u_1), ex (v_1,u_1). On a (en supposant l'interrupteur à gauche sur le ET introduit) :

$$\begin{array}{ll}
(u_3,u_1) \to^\star (u_1,u_3) & \text{(par récurrence)} \\
(u_1,u_3) \to (u_3,v_3) \to (v_3,u_3) \to (u_3,u_2) & \\
(u_3,u_2) \to^\star (u_2,u_3) & \text{(par récurrence)} \\
(u_2,u_3) \to (u_3,u_1) &
\end{array}$$

Si l'interrupteur est mis à droite, le parcours $(u_3,v_3) \to (v_3,u_3)$ se fait à la fin, juste avant (u_3,u_1). Enfin, pour (R3), avec interrupteur à gauche, on avait, avant application de la règle, un long chemin de la forme :

$$\cdots \to (u_1,v_1) \to (v_1,u_1) \to \cdots \to (u_2,v_2) \to (v_2,u_2) \to \cdots$$

qui devient $\cdots \to (u_1,u_3) \to (u_3,v_3) \to (v_3,u_3) \to (u_3,u_1) \to \cdots \to (u_2,u_3) \to (u_3,u_2) \to \cdots$.

(C2) Pour le préréseau 2, avec $\overset{\text{AX}}{u}$, $\overset{\text{ET}_G}{v}$ et $\overset{\text{TER}}{w}$, on a deux cycles et non un seul long voyage (de même pour les préréseau 6 et 7) :

$$\overset{\overline{x}}{(v,u)} \to \overset{x}{(u,v)} \to \overset{x\,\text{ET}\,\overline{x}}{(v,w)} \to \overset{x\,\text{ET}\,\overline{x}}{(w,v)} \to \overset{\overline{x}}{(v,u)} \qquad \overset{x}{(v,u)} \to \overset{\overline{x}}{(u,v)} \to \overset{x}{(v,u)}\ .$$

Pour le préréseau 3, on crée deux cycles disjoints.

(C3) Par récurrence sur la taille du G-réseau. Si un préréseau ne contient ni noeuds ET ni noeuds OU, il ne peut qu'être une union disjointe d'axiomes (R1).

Dans le cas d'un G-réseau, cette union doit être réduite à un seul axiome (sinon il y aurait plusieurs cycles et donc pas de long voyage), et on a donc un réseau séquentiel, obtenu par une application de (R1). On suppose maintenant qu'au moins une arête conclusion est étiquetée par une formule non atomique. Si l'une de ces formules est un OU, alors $G = (S, A)$ est tel que A est de la forme

$$A'' \cup \{(u_1, u_3), (u_2, u_3), (u_3, v_3)\} \setminus \{(u_1, v_1), (u_2, v_2)\} \ .$$

On voit facilement que le long voyage dans A induit un long voyage dans A', d'où il résulte par récurrence que A' est un réseau séquentiel, donc A est un réseau séquentiel par application de (R2). Reste le cas difficile où toutes les conclusions non atomiques sont des ET. CQFD

Pour régler le cas difficile de (C3), il faut introduire un outil supplémentaire. On a d'abord les propriétés suivantes.

(D1) Soit G est G-réseau, et soit u un noeud ET, avec arêtes entrantes (v_1, u) et (v_2, u). Montrer que quelque soit le choix d'interrupteurs le long voyage correspondant est de la forme

$$(v_1, u) \ \to \ \cdots \ \to \ (u, v_2) \ \to \ \cdots \ \to \ (v_2, u) \ \to \ (u, v_1) \ \to \ \cdots$$

si l'interrupteur sur u est à gauche, et de la forme

$$(v_2, u) \ \to \ \cdots \ \to \ (u, v_1) \ \to \ \cdots \ \to \ (v_1, u) \ \to \ (u, v_2) \ \to \ \cdots$$

si l'interrupteur sur u est à droite.

(D2) Soit G un G-réseau, et soit u un noeud OU, avec arêtes entrantes (v_1, u) et (v_2, u). Montrer que quelque soit le choix d'interrupteurs le long voyage correspondant est de la forme

$$(v_1, u) \ \to \ \cdots \ \to \ (u, v_1) \ \to \ \cdots \ \to \ (v_2, u) \ \to \ (u, v_2) \ \to \ \cdots$$

si l'interrupteur sur u est à gauche, et de la forme

$$(v_2, u) \ \to \ \cdots \ \to \ (u, v_2) \ \to \ \cdots \ \to \ (v_1, u) \ \to \ (u, v_1) \ \to \ \cdots$$

si l'interrupteur sur u est à droite.

PREUVE. (D1) On se concentre sur le cas où l'interrupteur est à gauche sur u. Il faut montrer que partant de (v_1, u), (v_2, u) ne sera pas visité avant (u, v_2). Supposons que l'on ait un long voyage de la forme

$$(v_1, u) \ \to \ \cdots_1 \ \to \ (v_2, u) \ \to \ (u, v_1) \ \to \ \cdots_2 \ \to \ (u, v_2) \ \to \ \cdots_3 \ \to \ (v_1, u) \ .$$

Alors, le parcours suivant serait suivi en mettant l'interrupteur sur u à droite (en gardant tous les autres dans leur position antérieure) :

$$(v_1, u) \rightarrow (u, v_2) \rightarrow \cdots_3 \rightarrow (v_1, u) \ ,$$

qui n'est pas un long voyage : puisque $\cdots_1 \rightarrow (v_2, u) \rightarrow (u, v_1) \rightarrow \cdots_2$ a disparu, les arêtes (v_2, u) et (u, v_1) ne sont pas parcourues.

(D2) Argument similaire. si l'on avait un voyage

$$(v_1, u) \rightarrow \cdots_1 \rightarrow (v_2, u) \rightarrow (u, v_2) \rightarrow \cdots_2 \rightarrow (u, v_1) \rightarrow \cdots_3 \rightarrow (v_1, u) \ ,$$

on aurait après changement de l'interrupteur en u le parcours

$$(v_1, u) \rightarrow (u, v_1) \rightarrow \cdots_3 \rightarrow (v_1, u) \ ,$$

qui n'est pas un long voyage. CQFD

On définit maintenant, pour toute arête $\overset{A}{(v, u)}$ l'*empire* $e(u, v)$ comme l'ensemble des arêtes (v', u') telles que pour tout choix d'interrupteurs le parcours $(u, v) \rightarrow \cdots \rightarrow (v, u)$ passe par (u', v') et (v', u'). Cette définition ne nous intéresse en fait que pour des arêtes de la forme $(v, \overset{\text{ET}}{u})$.
Dans tout ce qui suit, on simplifie les notations en identifiant les arêtes avec leur étiquette. On note donc, par exemple, eA pour $e(u, v)$ si A est l'étiquette de (u, v). Comme dans un long voyage on traverse la même arête une fois dans chaque sens, on précisera le sens à l'aide d'une flèche verticale montante ou descendante (cf. question (A)). Ainsi, si $\overset{A}{(v_1, u)}$ et $\overset{B}{(v_2, u)}$ sont les arêtes descendantes (i.e., entrantes) vers un noeud ET, on écrira le long voyage

$$(v_1, u) \rightarrow \cdots \rightarrow (u, v_2) \rightarrow \cdots \rightarrow (v_2, u) \rightarrow (u, v_1) \rightarrow \cdots$$

sous la forme

$${}^{A\downarrow}\rightarrow \ \ldots \ {}^{B\uparrow}\rightarrow \ \ldots \ \rightarrow {}^{B\downarrow}\rightarrow {}^{A\uparrow}\rightarrow \ \ldots \ .$$

(E1) Montrer que $A \in eA$ pour tout A, et que $eA \cap eB = \emptyset$ pour tout noeud ET d'arêtes entrantes A et B.

(E2) Montrer que eA vérifie les propriétés de clôture suivantes :

$$(e_1) \quad \frac{\overset{x}{(u,v_1)} \quad \overset{\text{AX}}{u} \quad \overset{\overline{x}}{(u,v_2)}}{(x \in eA) \Rightarrow (\overline{x} \in eA)}$$

$$(e_2) \quad \frac{\overset{C}{(v_1,u)} \quad \overset{\text{ET}}{u} \quad \overset{D}{(v_2,u)}}{((C\,\mathrm{ET}\,D) \in eA) \Rightarrow (C \in eA \text{ et } D \in eA)}$$

$$(e_3) \quad \frac{\overset{C}{(v_1,u)} \quad \overset{\text{OU}}{u} \quad \overset{D}{(v_2,u)}}{((C\,\mathrm{OU}\,D) \in eA) \Rightarrow (C \in eA \text{ et } D \in eA)}$$

$$(e_4) \quad \frac{\overset{C}{(v_1,u)} \quad \overset{\text{ET}}{u} \quad \overset{D}{(v_2,u)}}{((C \in eA) \Rightarrow ((C\,\mathrm{ET}\,D) \in eA)) \text{ et } ((D \in eA) \Rightarrow ((C\,\mathrm{ET}\,D) \in eA))}$$

Preuve. (E1) Le premier point est évident. On va montrer que le second est conséquence de (D1). Fixons un choix d'interrupteurs. Si $C \in eA$, alors ${}^{C\uparrow}$ et ${}^{C\downarrow}$ apparaissent le long de la portion ${}^{A\uparrow} \to \cdots \to {}^{A\downarrow}$ du long voyage, qui est disjointe de la portion ${}^{B\uparrow} \to \cdots \to {}^{B\downarrow}$.

(E2) La règle (e_1) est claire, car ${}^{\overline{x}\downarrow}$ (${}^{x\downarrow}$) suit toujours immédiatement ${}^{x\uparrow}$ (${}^{\overline{x}\uparrow}$). Pour les autres règles, on fixe toujours un choix d'interrupteurs.

- Règle (e_2). L'interrupteur sur le noeud u est, disons à gauche. Etant donné que $(C\,\mathrm{ET}\,D) \in eA$, on déduit comme ci-dessus que ${}^{C\downarrow}$ et ${}^{D\uparrow}$ sont visités durant le parcours ${}^{A\uparrow} \to \cdots \to {}^{A\downarrow}$. Supposons que ${}^{D\downarrow} \to {}^{C\uparrow}$ n'est pas dans ce parcours. Alors ${}^{A\uparrow}$ (${}^{A\downarrow}$) doit être entre ${}^{C\uparrow}$ et ${}^{C\downarrow}$ (entre ${}^{D\uparrow}$ et ${}^{D\downarrow}$), i.e., on a :

  $${}^{C\uparrow} \to \cdots_1 \to {}^{C\downarrow} \to {}^{(C\,\mathrm{ET}\,D)\downarrow} \to \cdots_2 \to {}^{(C\,\mathrm{ET}\,D)\uparrow} \to {}^{D\uparrow} \to \cdots_3 \to {}^{D\downarrow} \to {}^{C\uparrow}\,,$$

  avec ${}^{A\uparrow}$ (${}^{A\downarrow}$) dans $\cdots_1$ ($\cdots_3$). Si l'on change l'interrupteur sur u, le nouveau long voyage est

  $${}^{C\uparrow} \to \cdots_1 \to {}^{C\downarrow} \to {}^{D\uparrow} \to \cdots_3 \to {}^{D\downarrow} \to {}^{(C\,\mathrm{ET}\,D)\downarrow} \to \cdots_2 \to {}^{(C\,\mathrm{ET}\,D)\uparrow} \to {}^{C\uparrow}\;.$$

  Dans ce voyage, la portion ${}^{A\uparrow} \to \cdots \to {}^{A\downarrow}$ ne contient pas $(C\,\mathrm{ET}\,D)$, ce qui contredit l'hypothèse $((C\,\mathrm{ET}\,D) \in eA)$.

- Règle (e_3). Ce cas est semblable au précédent. On suppose de même que l'interrupteur sur u est à gauche. Et l'on arrive, mutatis mutandis, à

$${}^{C\uparrow} \to \ \cdots_1 \ \to {}^{D\downarrow} \to {}^{D\uparrow} \to \ \cdots_2 \ \to {}^{C\downarrow} \to {}^{(C\,\mathrm{OU}\,D)\downarrow} \to \ \cdots_3 \ \to {}^{(C\,\mathrm{OU}\,D)\uparrow} \to {}^{C\uparrow} \ ,$$

  avec ${}^{A\uparrow}$ (${}^{A\downarrow}$) dans $\cdots_2$ ($\cdots_1$). En changeant l'interrupteur sur u, on obtient

$${}^{A\uparrow} \to \ \cdots \ \to {}^{C\downarrow} \to {}^{C\uparrow} \to \ \cdots \ \to {}^{A\downarrow} \ ,$$

  ce qui contredit $((C\,\mathrm{OU}\,D) \in eA)$.
- Règle (e_4). C'est encore le même style. Soit donc toujours l'interrupteur sur u à gauche. On regarde d'abord $((C \in eA) \Rightarrow ((C\,\mathrm{ET}\,D) \in eA))$. En reprenant les $\cdots_i$ $(i = 1, 2, 3)$ du cas (e_2), on arrive à ${}^{A\uparrow}$ dans $\cdots_3$ et ${}^{A\downarrow}$ dans $\cdots_2$, et en changeant l'interrupteur on obtient que ${}^{A\uparrow} \to \ \cdots \ \to {}^{A\downarrow}$ ne contient pas C, ce qui contredit $C \in eA$. Pour $((D \in eA) \Rightarrow ((C\,\mathrm{ET}\,D) \in eA))$, on se ramène de même à ${}^{A\uparrow}$ dans $\cdots_2$ et ${}^{A\downarrow}$ dans $\cdots_1$.

On peut maintenant capitaliser ces préliminaires techniques.

(F1) Montrer qu'il existe des choix d'interrupteurs pour lesquels eA est l'ensemble des formules parcourues de ${}^{A\uparrow}$ à ${}^{A\downarrow}$.

(F2) Achever la preuve de (C3).

PREUVE. (F1) Par définition eA est toujours inclus dans cet ensemble, quelque soit le choix d'interrupteurs. Il suffit pour la réciproque de choisir les interrupteurs de sorte à ne jamais sortir de eA, i.e., si $(u, v) \in eA$ et $(u, v) \to (v, w)$, alors $(v, w) \in eA$. Si (u, v) est tel que (v, u) est arête sortante d'un noeud AX (resp. ET , OU), alors $(v, w) \in eA$ résulte de (e_1) (resp. (e_2), (e_3)). Si (u, v) est une arête entrante d'un noeud ET , alors, par définition de $\to$, soit (v, w) est l'arête sortante, soit (w, v) est l'autre arête entrante. Dans le premier cas, on conclut grâce à (e_4), et dans le second grâce à (e_4) et (e_2). Enfin, si (u, v) est arête entrante d'un noeud OU et si l'autre arête entrante n'est pas dans eA, alors on place l'interrupteur de sorte à ce que le voyage rebrousse chemin, et l'on a bien alors aussi $(v, w) \in eA$ puisque $(v, w) = (v, u)$ et que par définiton de eA on a $(u, v) \in eA$ si et seulement si $(v, u) \in eA$. Au total, on a choisi librement tous les interrupteurs, sauf ceux pour les noeuds OU ayant une seule arête entrante dans eA.

(F2) On reprend le cas laissé en suspens, à savoir celui d'un G-réseau G dont toutes les arêtes conclusion sortent d'un noeud ET . On choisit parmi celles-ci une arête ${}^{A\,\mathrm{ET}\,B}$ telle que $eA \cup eB$ soit maximal (pour l'inclusion). On va montrer :

(P) Toute arête autre que ${}^{A\,\mathrm{ET}\,B}$ est soit dans eA, soit dans eB.

On va même montrer que (P) est équivalente à la propriété suivante :

(Q) Il n'y a pas de noeud OU avec une arête entrante dans eA et l'autre arête entrante dans eB.

- (P)⇒(Q). L'arête sortante doit être dans eA ou eB, ce qui implique par (e_3) que les deux arêtes entrantes sont toutes deux dans eA ou toutes deux non dans eA : contradiction.
- (Q)⇒(P). La propriété (Q) garantit qu'il n'y a pas de conflit dans les choix d'interrupteurs pour réaliser (F1) pour eA et eB, i.e., il existe un choix d'interrupteurs tel que le voyage soit (avec abus de notation) :

  $$eA \;\rightarrow {}^{(A\,\mathrm{ET}\,B)\downarrow} \rightarrow {}^{(A\,\mathrm{ET}\,B)\uparrow} \rightarrow \; eB \;\rightarrow {}^{A\uparrow} \;\; .$$

  Comme nous sommes dans un G-réseau, ce voyage est un long voyage, et épuise donc toutes les arêtes.
- Preuve de (Q). Supposons qu'il y ait un tel noeud : $(\overset{C}{v_1}, u)$, $\overset{\mathrm{OU}}{u}$, $(\overset{D}{v_2}, u)$, $C \in eA$ et $D \in eB$. Soit ${}^{E\,\mathrm{ET}\,F}$ l'arête terminale en dessous de u, avec, disons u au-dessus de ^{F}. On peut faire un choix d'interrupteurs tel que eF soit

  $${}^{F\uparrow}\!\rightarrow \;\cdots_1 \;\rightarrow^{(C\,\mathrm{OU}\,D)\uparrow}\!\rightarrow^{C\uparrow}\!\rightarrow \;\cdots_2 \;\rightarrow^{D\downarrow}\!\rightarrow^{D\uparrow}\!\rightarrow \;\cdots_3 \;\rightarrow^{C\downarrow}\!\rightarrow^{(C\,\mathrm{OU}\,D)\downarrow}\!\rightarrow \;\cdots_4 \;\rightarrow^{F\downarrow} .$$

  En effet, la discipline de (F1) nous laisse assez de libertés pour faire un choix d'interrupteurs de sorte que le voyage à partir de ${}^{F\uparrow}$ parcoure successivement les sous-termes de F dont, disons C, est sous-terme. Et il est facile de voir par (D1) et (D2) (employés récursivement) que ce voyage doit se conformer à une discipline de pile, i.e., que ces sous-termes successifs sont ensuite retraversés en sens inverse et en ordre inverse (d'où la portion $\rightarrow\ \cdots_4$).
  On utilise maintenant les hypothèses $C \in eA$ et $D \in eB$, qui impliquent a fortiori que D apparaît dans les deux sens dans ${}^{B\uparrow} \rightarrow \;\cdots\; \rightarrow {}^{B\downarrow}$ et que C n'apparaît dans aucun sens dans cette même portion de voyage. Ceci ne laisse pas de choix quand au placement de ${}^{B\uparrow}$ et de ${}^{B\downarrow}$, qui doivent respectivement être dans $\cdots_2$ et $\cdots_3$. On en déduit

  $$eB \;\subseteq\; \left({}^{B\uparrow} \rightarrow \;\cdots\; \rightarrow {}^{B\downarrow}\right) \;\subseteq\; \left({}^{F\uparrow} \rightarrow \;\cdots\; \rightarrow {}^{F\downarrow}\right) \;=\; eF \; .$$

  De même, on a aussi $eA \subseteq eF$. Ceci contredit la maximalité, car $eF \cup eG$ contient alors strictement $eA \cup eB$.

On peut enfin terminer la preuve de (C3). On prouve d'abord que eA détermine un préréseau. Les règles (e_1) à (e_4) le garantissent, à moins qu'il n'y ait un noeud OU possédant une seule arête entrante dans eA. Ceci est impossible par (Q). Donc eA détermine un préréseau. Il est facile de voir que eA est un G-réseau (les voyages dans eA correspondent aux parcours ${}^{A\uparrow} \to \cdots \to {}^{A\downarrow}$ dans G), et on peut alors conclure par récurrence : eA et eB sont des réseaux séquentiels disjoints, et G est obtenu à partir de eA et eB par la règle (R2). CQFD

16 Lambda-calcul et machine de Krivine

Source. P.-L. Curien, “An abstract framework for environment machines”, Theoretical Computer Science 82 (1991).

Le λ-calcul est une abstraction du mécanisme de déclaration et d’appel de procédure. La syntaxe est donnée par :

$$M ::= x \mid MM \mid \lambda x.M\ .$$

x est un variable, $\lambda x.M$ est une abstraction (ou fonction, ou procédure), et MN désigne l’application de M à N. En particulier $(\lambda x.M)N$ représente un appel de procédure. D’où la règle de calcul :

$$(\beta) \quad (\lambda x.M)N \ \rightarrow \ M\{N/x\}$$

qui remplace le paramètre formel par le paramètre réel dans le corps M de la procédure. Comme les dx dans les intégrales ou les $\forall x$ en logique, la variable x est liée dans M, et on identifie par exemple $\lambda x.x$ et $\lambda y.y$. On utilise aussi la notation (let $x = N$ in M) pour $(\lambda x.M)N$.

Pour rendre le langage moins abstrait, on y ajoute des constantes n (pour tout entier naturel). Ainsi, $(\lambda x.x)4 \ \rightarrow \ 4$. On ajoute aussi l’addition $+$ (on écrira $(+M)N$ pour l’addition de M et N). La syntaxe est donc finalement :

$$M ::= x \mid MM \mid \lambda x.M \mid n \mid +\ .$$

Dans un terme comme $((\lambda u.(\lambda f.f5))3)((\lambda x.x)(\lambda y.y))$, deux réductions sont possibles : celles des sous-termes $(\lambda x.x)(\lambda y.y)$ et $(\lambda u.(\lambda f.f5))3$, respectivement. On fixe une stratégie de calcul (appel par nom), qui dit dans quel ordre faire ces calculs :

$$(R_1) \ \frac{M \rightarrow^\star \lambda x.P}{MN \ \rightarrow \ P\{N/x\}} \qquad (R_2) \ \frac{M \rightarrow^\star m \quad N \rightarrow^\star n}{M+N \ \rightarrow \ m+n}$$

(dans les hypothèses des règles, $\rightarrow^\star$ désigne la clôture réflexive et transitive de $\rightarrow$).

(A1) Montrer le calcul de $((\lambda u.(\lambda f.f5))3)((\lambda x.x)(\lambda y.y))$ par la règle (R_1).

(A2) Que peut-on dire de $(\lambda x.xx)(\lambda y.yy)$, et de $\lambda f.((\lambda x.f(xx))\ (\lambda x.f(xx)))$?

PREUVE. (A1)

$$\dfrac{\dfrac{\overset{\text{(réflexivité)}}{\lambda u.(\lambda f.f5) \to^\star \lambda u.(\lambda f.f5)}}{(\lambda u.(\lambda f.f5))3 \ \to\ \lambda f.f5}}{((\lambda u.(\lambda f.f5))3)((\lambda x.x)(\lambda y.y)) \ \to\ ((\lambda x.x)(\lambda y.y))5} \qquad \dfrac{\dfrac{\lambda x.x \to^\star \lambda x.x}{(\lambda x.x)(\lambda y.y) \ \to\ \lambda y.y}}{((\lambda x.x)(\lambda y.y))5 \ \to\ 5}$$

(A2) On a $(\lambda x.xx)(\lambda y.yy) \ \to\ (\lambda y.yy)(\lambda y.yy)$, donc ce terme se réduit (indéfiniment) sur lui-même. Le terme $Y = \lambda f.((\lambda x.f(xx))\ (\lambda x.f(xx)))$ est tel que

$$YM \ \to\ (\lambda x.M(xx))(\lambda x.M(xx)) \ \to\ M((\lambda x.M(xx))(\lambda x.M(xx)))\ .$$

Donc $(\lambda x.M(xx))(\lambda x.M(xx))$ est un point fixe de M. Ici aussi le calcul est infini, mais produit des M à chaque étape. CQFD

On veut maintenant décrire une implantation de (R_1) et (R_2) à l'aide d'une pile. La règle (R_1) dit en effet que pour évaluer MN, il faut d'abord évaluer M, et empiler N. Par ailleurs, une substitution peut être coûteuse, d'où l'idée de maintenir une fonction E associant des termes aux variables, que l'on représente par des paires N/x et que l'on nomme un environnement. On ne procède alors aux substitutions effectives que lorsqu'elles sont nécessaires pour poursuivre le calcul. D'où (en première approximation) des états de la forme (terme,environnement,pile), ou (M, E, S), S étant une liste de termes. Notation : si E est un environnement, $E[M/x]$ désigne la fonction coïncidant avec E sauf éventuellement en x où elle vaut M.

(B1) Ecrire des règles de la forme $(M, E, S) \ \to\ (M', E', S')$ correspondant à chaque cas de M (application, abstraction, variable,...). Une telle formalisation d'un modèle d'exécution est appelée une machine abstraite.

(B2) Critique de ces règles. Analyser l'exécution de

$$\text{let } x = 4 \text{ in } (\text{let } f = (\lambda y.(+x)y) \text{ in } (\text{let } x = 3 \text{ in } f6))\ .$$

PREUVE. On utilise la notation :: pour l'ajout d'un élément à la pile.

(B1) La machine :

$$\begin{array}{lll}
(MN, E, S) & \to & (M, E, (N :: S)) \\
(\lambda x.M, E, (N :: S)) & \to & (M, E[N/x], S) \\
(x, E, S) & \to & (N, E, S) \ \ ((N/x) \in E) \\
(+, E, (M :: S)) & \to & (M, E, (+_g :: S)) \\
(m, E, (+_g :: N :: S)) & \to & (N, E, (+_d :: m :: S)) \\
(n, E, (+_d :: m :: S)) & \to & (m + n, E, S)
\end{array}$$

(B2) Le terme de l'énoncé est $(\lambda x.(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y))4$ et s'évalue par (R_1) et (R_2) comme suit :

$$\begin{aligned}
(\lambda x.(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y))4 &\rightarrow (\lambda f.(\lambda x.(f6))3)(\lambda y.(+4)y)\\
&\rightarrow (\lambda x.(\lambda y.(+4)y)6)3\\
&\rightarrow (\lambda y.(+4)y)6\\
&\rightarrow 4+6\\
&\rightarrow 10\ .
\end{aligned}$$

Par contre la machine donne :

$(\lambda x.(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y))4$		NIL
$\lambda x.(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y)$		4
$(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y)$	$4/x$	NIL
$\lambda f.(\lambda x.(f6))3$	$4/x$	$\lambda y.(+x)y$
$(\lambda x.(f6))3$	$(\lambda y.(+x)y)/f\ ,\ 4/x$	NIL
$\lambda x.(f6)$	$(\lambda y.(+x)y)/f\ ,\ 4/x$	3
$f6$	$(\lambda y.(+x)y)/f\ ,\ 3/x$	NIL
f	$(\lambda y.(+x)y)/f\ ,\ 3/x$	6
$(\lambda y.(+x)y)$	$(\lambda y.(+x)y)/f\ ,\ 3/x$	6
$(+x)y$	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	NIL
$+x$	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	y
$+$	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	$x :: y$
x	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	$+_g :: y$
3	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	$+_g :: y$
y	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	$+_d :: 3$
6	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	$+_d :: 3$
9	$6/y\ ,\ (\lambda y.(+x)y)/f\ ,\ 3/x$	NIL

Le problème est que pour la règle (R_1) la liaison de x qui compte pour f est celle $(4/x)$ qui prévalait au moment de la déclaration de f, alors que la machine abstraite efface cette information au profit de la liaison $3/x$ prévalant au moment où f est utilisé. CQFD

Une machine correcte doit empiler des paires (terme,environnement) (notation $M[E]$)) de sorte à pouvoir toujours maintenir l'environnement de déclaration en le protégeant contre les redéfinitions dynamiques d'identificateurs. D'où la syntaxe suivante (noter les emboîtements d'environnements) :

$$E ::= \emptyset \mid E[M[E]/x]\ .$$

- **(C1)** Rectifier les règles de la machine avec cette nouvelle définition d'environnement. (Cette machine est connue sous le nom de machine de Krivine.) Retraiter l'exemple (B2).
- **(C2)** Décrire les états terminaux de la machine.
- **(C3)** (Dans cette question et la suivante, on se limite au λ-calcul pur, sans n ni $+$.) Associer à tout état (M, E, S) de la machine un terme $(M, E, S)^\circ$ du λ-calcul, de sorte que cette traduction préserve la réduction, i.e., si $(M, E, S) \to (M', E', S')$, alors $(M, E, S)^\circ \to^\star (M', E', S')^\circ$.
- **(C4)** Les questions (C2) et (C3) permettent-elles de conclure que la machine de Krivine fournit une implémentation complète des règles (R_1) et (R_2) ? (Donner un sens précis au mot "complet".)

Preuve. (C1) La machine :

$$\begin{array}{lcl}
(MN, E, S) & \to & (M, E, (N[E] :: S)) \\
(\lambda x.M, E, (N[E'] :: S)) & \to & (M, E[N[E']/x], S) \\
(x, E, S) & \to & (N, E', S) \quad ([N[E']/x] \in E) \\
 & & \\
(+, E, (M[E'] :: S)) & \to & (M, E', (+_g :: S)) \\
(m, E, (+_g :: (N[E'] :: S)) & \to & (N, E', (+_d :: m :: S)) \\
(n, E, (+_d :: m :: S)) & \to & (m + n, E, S)
\end{array}$$

L'exemple (_ désigne un environnement qu'il n'est pas utile d'expliciter) :

Terme	Environnement	Pile
$(\lambda x.(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y))4$		NIL
$\lambda x.(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y)$		$4[\,]$
$(\lambda f.(\lambda x.(f6))3)(\lambda y.(+x)y)$	$4[\,]/x$	NIL
$\lambda f.(\lambda x.(f6))3$	$4[\,]/x$	$(\lambda y.(+x)y)[4[\,]/x]$
$(\lambda x.(f6))3$	$(\lambda y.(+x)y)[4[\,]/x]/f$, $4[\,]/x$	NIL
$\lambda x.(f6)$	$(\lambda y.(+x)y)[4[\,]/x]/f$, $4[\,]/x$	$3[\_]$
$f6$	$(\lambda y.(+x)y)[4[\,]/x]/f$, $3[\_]/x$	NIL
f	$(\lambda y.(+x)y)[4[\,]/x]/f$, $3[\_]/x$	$6[\_]$
$(\lambda y.(+x)y)$	$4[\,]/x$	$6[\_]$

Le reste du calcul est sans surprise :

$(+x)y$	$6[\_]/y\ ,\ 4[\,]/x$	NIL
$+x$	$6[\_]/y\ ,\ 4[\,]/x$	$y[6[\_]/y, 4[\,]/x]$
$+$	$6[\_]/y\ ,\ 4[\,]/x$	$x[6[\_]/y, 4[\,]/x] :: y[6[\_]/y, 4[\,]/x]$
x	$6[\_]/y\ ,\ 4[\,]/x$	$+_g :: y[6[\_]/y, 4[\,]/x]$
4		$+_g :: y[6[\_]/y, 4[\,]/x]$
y	$[6[\_]/y\ ,\ 4[\,]/x$	$+_d :: 4$
6	$\_$	$+_d :: 4$
10	$\_$	NIL

(C2) La machine s'arrête dans les circonstances suivantes :

- L'état est $(\lambda x.M, E, \mathrm{NIL})$. Cela correspond au fait que l'expression calculée est une fonction : la machine s'arrête quand elle a reconnu qu'il s'agissait d'une proédure.
- L'état est (x, E, S), et E ne contient pas de liaison pour x. Cette situation ne se produit pas si l'on est parti d'un terme clos (i.e., sans variables libres ; voir plus bas).
- L'état est (m, E, NIL). C'est le résultat du calcul.
- L'état est $(+, E, \mathrm{NIL})$. Ce cas ressemble au premier : la machine a reconnu la fonction d'addition.
- L'état est $(m, E, +_g)$. La machine a reconnu la fonction $m + \_$.

Il y a d'autres cas de terminaison plus "sales", qu'un bon typage excluerait. Par exemple :

$$((+(\lambda x.x))4\ ,\ \emptyset\ ,\ \mathrm{NIL}) \quad \rightarrow^\star \quad (\lambda x.x\ ,\ \emptyset\ ,\ (+_g :: 4[\,]))$$

Si l'on se limite au λ-calcul pur, les seuls arrêts possibles sont les deux premiers ci-dessus.

(C3) On définit la traduction comme suit :

$$(M, E, (N_1 :: N_2 :: N_p :: \mathrm{NIL}))^\circ \quad = \quad (\cdots((M\{E^\circ\})N_1)\cdots)N_p$$

où $(N[E]/x, E')^\circ = ((N[E])^\circ/x, (E')^\circ)$ et $(N[E])^\circ = N\{E^\circ\}$. Dans ces équations, $\{\,\}$ dénote par rapport à $[\,]$ la substitution (simultanée) effectivement réalisée.

(C4) Si l'on part d'un terme clos et de l'état initial $(M, \emptyset, \mathrm{NIL})$, et si $(M, \emptyset, \mathrm{NIL})$ $\rightarrow^\star$ Ã(M', E', S'), on vérifie facilement par récurrence sur la longueur de la réduction que toutes les variables libres de M' sont liées dans E' : un arrêt sur (x, E, S) n'est donc pas possible, puisque la condition de bord de cette règle ne peut pas s'appliquer. Il ne reste donc plus (cf. (C2)) qu'un seul type d'état final :

$(\lambda x.P, E, \mathrm{NIL})$. On sait par (C3) que si $(M, \emptyset, \mathrm{NIL}) \to^{\star} (\lambda x.M', E', \mathrm{NIL})$, alors

$$M \to^{\star} ((\lambda x.M')\{(E')^{\circ}\}\ ,$$

qui est une abstraction, donc non réductible par (R_1). En d'autres termes, par (C2) et (C3), on sait que pour peu que la machine termine, elle arrive bien au résultat. Mais (C2) et (C3) ne garantissent pas que la machine termine si le calcul de M par (R_1) termine (sur une abstraction). Il faut un argument de plus. Si $(M, \emptyset, \mathrm{NIL}) \to^{\star} \cdots$ ne termine pas et si $M \to^{\star} \cdots$ termine, alors, par (C3), à partir d'un certain stade, les états de la machine se traduisent sur le même terme (sinon, on aurait une (R_1)-réduction infinie). Ceci implique que seules les règles $(MN, E, S) \to (M, E, (N[E] :: S))$ et $(x, E, S) \to (N, E', S)$ sont utilisées, qui font diminuer le terme ou l'environnement, et donc terminent (plus précisément, on utilise l'ordre lexicographique sur (taille(E),taille(N)). Contradiction. CQFD

17 Entiers naturels paresseux

Source. L. Colson, "About intensional behaviour of primitive recursive algorithms", Proceedings of ICALP 1989, Lecture Notes in Computer Science 372, Springer.

On rappelle que $\mathbf{N}$ désigne l'ensemble des entiers naturels $0, 1, \ldots$ On définit l'ensemble des entiers naturels paresseux par :

$$\mathbf{N}_\perp = \{S^n(\perp) \mid n \in \mathbf{N}\} \cup \{S^n(0) \mid n \in \mathbf{N}\} \cup \{S^\infty(\perp)\} \,.$$

$\mathbf{N}_\perp$ comprend trois sortes d'objets : les entiers naturels $S^n(0)$ ($n \in \mathbf{N}$) (par exemple, 3 est noté $S(S(S(0))) = S^3(0)$), les points $S^n(\perp)$ ($n \in \mathbf{N}$), et enfin un point dit infini $S^\infty(\perp)$ (tous les autres sont dits finis). Les points $S^m(\perp)$ et le point $S^\infty(\perp)$ sont appelés incomplets. L'idée d'un entier incomplet $S^m(\perp)$ est qu'il représente une information partielle : "si le calcul termine, le résultat sera $\geq m$ (i.e., $S^n(0)$ avec $n \geq m$)". Conformément à ces intuitions, on ordonne $\mathbf{N}_\perp$ comme suit : $x \leq y$ si et seulement si $x = S^m(\perp)$ et y est de la forme $y = S^m(z)$ (on convient que $S^0(\perp) = \perp$, $S^0(0) = 0$ et $S^\infty(\perp) = S^m(S^\infty(\perp))$ pour tout $m \in \mathbf{N}$). En particulier, $\perp$ est le plus petit élément (il est d'usage de l'appeler "bottom").

(A1) Quels sont les éléments maximaux de $\mathbf{N}_\perp$?

(A2) Montrer que dans $\mathbf{N}_\perp$ toute suite croissante a une borne supérieure.

(A3) La somme séparée de deux ensembles ordonnés D et E est définie comme suit : $D + E = \{\perp\} \cup \{(x.1) \mid x \in D\} \cup \{(y.2) \mid y \in E\}$, avec $z \leq z'$ si et seulement si $z = \perp$, ou ($z = x.1, z' = x'.1$ et $x \leq x'$) ou ($z = y.2, z' = y'.2$ et $y \leq y'$). Montrer que $\mathbf{N}_\perp$ et $\{0\} + \mathbf{N}_\perp$ sont isomorphes en tant que structures ordonnées. Justifier informellement en quoi $\mathbf{N}_\perp$ est obtenu comme une construction limite.

Preuve. (A1) Les éléments maximaux sont les $S^n(0)$ et $S^\infty(\perp)$.

(A2) Toutes les suites croissantes sont stationnaires sauf celles de la forme $S^{n_1}(\perp)$,…,$S^{n_i}(\perp)$,… (avec $n_1, \ldots, n_i, \ldots$ croissante au sens de $\mathbf{N}$), dont la borne supérieure est $S^\infty(\perp)$.

(A3) La bijection $i : (\{0\} + \mathbf{N}_\perp) \to \mathbf{N}_\perp$ est donnée par :

$$i(\perp) = \perp \quad i(0.1) = 0 \quad i(S^m(x).2) = S^{m+1}(x)$$

(en particulier, $i(1,\bot) = S(\bot)$). Il est facile de voir qu'en itérant la construction $\{0\} + \_$ à partir de $\emptyset$, on construit progressivement $\mathbf{N}_\bot$: au stade n, on a les $S^m(x)$ $(m \leq n)$. CQFD

On définit maintenant la syntaxe des fonctions primitives récursives (p.r.) :

$$\frac{}{0:0} \qquad \frac{}{S:1} \qquad \frac{}{\pi^n_i : n} \qquad \frac{g_1 : m \quad \cdots \quad g_n : m \quad f : n}{f \circ \langle g_1, \ldots, g_n \rangle : m} \qquad \frac{g : n \quad h : n+2}{rec(g,h) : n+1}$$

On écrit $f : n$ pour "f est p.r. d'arité n", et on dit que f est un *combinateur* p.r. Informellement, les combinateurs p.r. d'arité n représentent des fonctions à n arguments : 0 est une fonction constante, S est la fonction successeur, les fonctions π^n_i sont des fonctions de projection, $f \circ \langle g_1, \ldots, g_n \rangle$ représente la composition de fonctions ; enfin, $rec(g,h)$ représente la récursion primitive, qui donne la possibilité de définir une fonction par récurrence sur l'un de ses arguments. Puis on définit l'ensemble T des *termes* primitifs récursifs comme suit ($\bot$, contrepartie syntaxique du $\bot$ sémantique, désigne l'indéfini) :

$$\frac{f : n \quad t_1 \in T \quad \cdots \quad t_n \in T}{f(t_1, \ldots, t_n) \in T} \qquad \frac{}{\bot \in T.}$$

D'habitude, on interprète cette syntaxe dans $\mathbf{N}$. Ici, on étend l'interprétation à $\mathbf{N}_\bot$, i.e., à $f : n$ on associe une fonction $\underline{f} : (\mathbf{N}_\bot)^n \to \mathbf{N}_\bot$ (c'est ce qu'on appelle une sémantique dénotationnelle). Et en même temps, on définit une fonction $\overline{f} : (\mathbf{N}_\bot)^n \to \{1, \ldots, n\}$ (pour $n \geq 1$), dont l'utilité va bientôt apparaître.

$$\frac{}{\underline{0} = 0}$$

$$\frac{}{\underline{S}(x) = S(x)} \qquad \frac{}{\overline{S}(x) = 1}$$

$$\frac{}{\underline{\pi^n_i}(x_1, \ldots, x_n) = x_i} \qquad \frac{}{\overline{\pi^n_i}(x_1, \ldots, x_n) = i}$$

$$\frac{}{\underline{f \circ \langle g_1, \ldots, g_n \rangle}(\vec{x}) = \underline{f}(\underline{g_1}(\vec{x}), \ldots, \underline{g_n}(\vec{x}))} \qquad \frac{\overline{f}(\underline{g_1}(\vec{x}), \ldots \underline{g_n}(\vec{x})) = j}{\overline{f \circ \langle g_1, \ldots, g_n \rangle}(\vec{x}) = \overline{g_j}(\vec{x})}$$

$$\frac{}{\underline{rec(g,h)}(\bot,\vec{y})=\bot} \qquad \frac{}{\overline{rec(g,h)}(\bot,\vec{y})=1}$$

$$\frac{}{\underline{rec(g,h)}(0,\vec{y})=\underline{g}(\vec{y})} \qquad \frac{\overline{g}(\vec{y})=i}{\overline{rec(g,h)}(0,\vec{y})=i+1}$$

$$\frac{}{\underline{rec(g,h)}(S(x),\vec{y})=\underline{h}(x,\underline{rec(g,h)}(x,\vec{y}),\vec{y})} \qquad \frac{\overline{h}(x,\underline{rec(g,h)}(x,\vec{y}),\vec{y})=2}{\overline{rec(g,h)}(S(x),\vec{y})=\overline{rec(g,h)}(x,\vec{y})}$$

$$\frac{\overline{h}(x,\underline{rec(g,h)}(x,\vec{y}),\vec{y})=j\geq 3}{\overline{rec(g,h)}(S(x),\vec{y})=j-1} \qquad \frac{\overline{h}(x,\underline{rec(g,h)}(x,\vec{y}),\vec{y})=1}{\overline{rec(g,h)}(S(x),\vec{y})=1}$$

Les termes p.r. sont interprétés comme suit :

$$\underline{f(t_1,\dots,t_n)}=\underline{f}(\underline{t_1},\dots,\underline{t_n}) \qquad \underline{\bot}=\bot\ .$$

(B1) Ecrire un combinateur d'addition à gauche add_g (i.e., dont la définition exploite une récurrence sur le premier argument). Quelle est la valeur de $\underline{add_g}(S^m(\bot),S^n(\bot))$ ("comportement intensionnel" de add_g) ? Ecrire le combinateur d'addition à droite add_d. A-t-on $\underline{add_d}=\underline{add_g}$ sur $(\mathbf{N}_\bot)^2$?

(B2) Montrer, pour tout combinateur f, que $\underline{f}$ est bien définie et à valeur finie sur tout vecteur fini (i.e., dont toutes les composantes sont finies), et est croissante (on définit $(x_1,\dots,x_n)\leq(y_1,\dots,y_n)$ par $x_1\leq y_1,\dots,x_n\leq y_n$). Etendre la définition de $\underline{f}$ à $(\mathbf{N}_\bot)^n$, i.e., définir $\underline{f}$ sur des vecteurs ayant au moins une composante égale à $S^\infty(\bot)$. Montrer que cette extension est continue en ce sens que si $\underline{f}(\vec{x})=S^\infty(\bot)$, alors pour tout m il existe un vecteur fini $\vec{d}\leq\vec{x}$ tel que $\underline{f}(\vec{d})\geq S^m(\bot)$.

(B3) Montrer que si $\overline{f}(\vec{x})=i$ alors

$$(\vec{x'}\geq\vec{x}\ \text{et}\ z_i=x_i)\quad\Rightarrow\quad(\underline{f}(\vec{x'})=\underline{f}(\vec{x})\ \text{et}\ \overline{f}(\vec{x'})=\overline{f}(\vec{x}))\ .$$

La fonction $\overline{f}$ indique donc quel argument "bloque" le calcul.

(B4) Montrer que si $\underline{f}(x_1,\dots,x_n)$ est incomplet et si $\overline{f}(x_1,\dots,x_n)=i$, alors x_i est incomplet.

PREUVE. (B1) On définit add_g et add_d comme suit :

$$\begin{array}{ll} add_g(0,y) = y & add_d(x,0) = x \\ add_g(S(x),y) = S(add_g(x,y)) & add_d(x,S(y) = S(add_d(x,y)) \end{array}$$

Formellement : $add_g = rec(\pi_1^1, S \circ \pi_2^3)$ et $add_d = add_g \circ \langle \pi_2^2, \pi_1^2 \rangle$. L'addition à gauche récurre sur le premier argument, et le résultat garde la trace du nombre de dépliages de la définition récursive :

$$\begin{array}{rclcl} add_g(S^m(\bot),x) & = & S(add_g(S^{m-1}(\bot),x) & = & \cdots \\ & = & S^m(add_g(\bot,x)) & = & S^m(\bot)\ . \end{array}$$

En particulier, $add_g \neq add_d$, car

$$add_g(S^m(\bot),S^n(\bot)) = S^m(\bot) \quad \text{et} \quad add_d(S^m(\bot),S^n(\bot)) = S^n(\bot)\ .$$

(B2) On utilise la récurrence suivante, qu'on appellera récurrence structurelle : sur la définition de fonction p.r., puis sur le nombre de S du premier argument dans le cas rec. Par exemple, montrons que $rec(g,h)(S(x),\vec{y})$ est bien défini et est fini. On sait par récurrence que h est à valeur finie sur tout vecteur fini. On sait aussi par récurrence que $rec(g,h)(x,\vec{y})$ est bien défini et est fini. Donc $(x, rec(g,h)(x,\vec{y}),\vec{y})$ est un vecteur fini, d'où la conclusion.
La démonstration de ce que f est croissante est sans surprise, par cette même récurrence (noter que si $\vec{y} \leq \vec{z}$, alors on a $rec(g,h)(\bot,\vec{y}) = \bot \leq rec(g,h)(x,\vec{z})$, pour tout x). L'extension de $\underline{f}$ est définie par passage à la limite :

$$\underline{f}(\vec{x}) = \vee\{\underline{f}(\vec{d}) \mid \vec{d} \text{ est fini et } \vec{d} \leq \vec{x}\}\ .$$

On omet la vérification de la continuité, sans surprise non plus.

(B3) Par récurrence structurelle. Les cas $f = S$ et $f = \pi_i^n$ sont évidents. Pour la composition, on a par récurrence

$$\begin{array}{l} \underline{g_j}(\vec{x'}) = \underline{g_j}(\vec{x}),\ \ \underline{f}(\underline{g_1}(\vec{x'}),\ldots,\underline{g_j}(\vec{x}),\ldots,\underline{g_n}(\vec{x'})) = \underline{f}(\underline{g_1}(\vec{x}),\ldots,\underline{g_j}(\vec{x}),\ldots,\underline{g_n}(\vec{x})) \\ \overline{g_j}(\vec{x'}) = \overline{g_j}(\vec{x}),\ \ \overline{f}(\underline{g_1}(\vec{x'}),\ldots,\underline{g_j}(\vec{x}),\ldots,\underline{g_n}(\vec{x'})) = \overline{f}(\underline{g_1}(\vec{x}),\ldots,\underline{g_j}(\vec{x}),\ldots,\underline{g_n}(\vec{x}))\ , \end{array}$$

qui donnent $(\underline{f \circ \langle g_1,\ldots,g_n\rangle})(\vec{x'}) = (\underline{f \circ \langle g_1,\ldots,g_n\rangle})(\vec{x})$ et $(\overline{f \circ \langle g_1,\ldots,g_n\rangle})(\vec{x'})$ $= (\overline{f \circ \langle g_1,\ldots,g_n\rangle})(\vec{x})$. On considère ensuite $rec(g,h)$ en $(\bot,\vec{y})$: on a alors $i = 1$ et on a bien

$$\underline{rec(g,h)}(\bot,\vec{y'}) = \bot = \underline{rec(g,h)}(\bot,\vec{y}) \quad \overline{rec(g,h)}(\bot,\vec{y'}) = 1 = \overline{rec(g,h)}(\bot,\vec{y})\ ,$$

pour tout $\vec{y'} \geq \vec{y}$. Pour $rec(g,h)$ en $(0,\vec{y})$, on a :

$$\underline{rec(g,h)}(0,\vec{y'}) = \underline{g}(\vec{y'}) = \underline{g}(\vec{y}) = \underline{rec(g,h)}(0,\vec{y})$$

$$\overline{rec(g,h)}(0,\vec{y'}) = \overline{g}(\vec{y'}) + 1 = \overline{g}(\vec{y}) + 1 = \overline{rec(g,h)}(0,\vec{y'})$$

(on remarque qu'un $n+1$-uplet $\geq (0,\vec{y})$ est de la forme $(0,\vec{y'})$). Il reste $rec(g,h)$ en $(S(x),\vec{y})$. Un vecteur satisfaisant les conditions de l'énoncé s'écrit $(S(x'),\vec{y'})$, et est tel que $(x',\vec{y'}) \geq (x,\vec{y})$, en laissant la même composante invariante. Soit $j = \overline{h}(x, \underline{rec(g,h)}(x,\vec{y}), \vec{y})$. On traite seulement le cas $j = 2$ (les autres sont plus simples). On a $\underline{rec(g,h)}(x',\vec{y'}) = \underline{rec(g,h)}(x,\vec{y})$ et $\overline{rec(g,h)}(x',\vec{y'}) = \overline{rec(g,h)}(x,\vec{y})$ par récurrence appliquée à x. Par récurrence appliquée à h, on a alors

$$\overline{h}(x', \underline{rec(g,h)}(x',\vec{y'}), \vec{y'}) = \overline{h}(x', \underline{rec(g,h)}(x,\vec{y}), \vec{y'}) = 2\ .$$

Donc

$$\overline{rec(g,h)}(S(x'),\vec{y'}) = \overline{rec(g,h)}(x',\vec{y'}) = \overline{rec(g,h)}(x,\vec{y}) = \overline{rec(g,h)}(S(x),\vec{y})\ .$$

Et on a aussi $\underline{rec(g,h)}(S(x'),\vec{y'}) = \underline{h}(x, \underline{rec(g,h)}(x,\vec{y}), \vec{y}) = \underline{rec(g,h)}(S(x),\vec{y})$.

(B4) Par récurrence structurelle (pas de difficultés). Cette propriété dit que pour faire croître le résultat (incomplet), il faut faire croître le i-ème argument (incomplet). CQFD

Une autre façon de décrire la signification des termes p.r. est de donner des règles de calcul (c'est ce qu'on appelle une sémantique opérationnelle).

$$\begin{array}{rcl}
\pi_i^n(t_1,\ldots,t_n) & \to & t_i \\
(f \circ \langle\vec{g}\rangle)(\vec{t}) & \to & f(g_1(\vec{t}),\ldots,g_n(\vec{t})) \\
rec(g,h)(\bot,\vec{t}) & \to & \bot \\
rec(g,h)(0,\vec{t}) & \to & g(\vec{t}) \\
rec(g,h)(S(s),\vec{t}) & \to & h(s, rec(g,h)(s,\vec{t}), \vec{t})
\end{array}$$

$$\frac{s_1 \to s_2}{S(s_1) \to S(s_2)} \qquad \frac{s_1 \to s_2}{rec(g,h)(s_1,\vec{t}) \to rec(g,h)(s_2,\vec{t})}$$

Ce système de règles est déterministe, i.e., il n'y a pas de choix dans l'application d'une règle. L'avant-dernière règle revient à donner un S en output et à continuer le calcul. La dernière règle s'applique lorsque le premier argument n'est pas encore assez calculé pour faire apparaître qu'il est de la forme $\bot$, 0 ou $S(x)$.

Le but de la question suivante est de relier les deux sémantiques (dénotationnelle et opérationnelle). Pour tout terme $t \in T$, on note $\text{coût}_{\to}(t)$ la longueur de la réduction de t jusqu'à un terme de la forme $S^m(\bot)$ ou $S^m(0)$ (forme normale), on note $\text{coût}_S(t)$ cet m, et on note $\text{coût}(t) = \text{coût}_{\to}(t) + \text{coût}_S(t)$. On remarque :

$$s \to t \;\Rightarrow\; (\text{coût}_{\to}(s) = \text{coût}_{\to}(t) + 1) \text{ et } (\text{coût}_S(s) = \text{coût}_S(t)) \; .$$

(C1) Montrer que $\to$ termine toujours, et s'achève toujours sur un terme de la forme $S^m(\bot)$ ou $S^m(0)$ (ce qui justifie la définition de $\text{coût}(s)$).

(C2) Montrer que si $s \to t$, alors $\underline{s} = \underline{t}$.

(C3) Montrer que $t \to^\star \underline{t}$, pour tout terme t.

(C4) Montrer l'implication suivante (pour tout terme p.r.) :

$$(\overline{f}(\underline{t_1}, \ldots, \underline{t_n}) = i) \Rightarrow (\text{coût}(t_i) \leq \text{coût}(f(t_1, \ldots, t_n))) \; .$$

Preuve. (C1) On montre en même temps que le calcul termine, et qu'il termine sur un terme de la forme $S^m(\bot)$ ou $S^m(0)$ (ce que l'on appellera bien terminer). La preuve se fait en deux étapes. On montre d'abord par récurrence structurelle pour tout f que si $t_1, \ldots, t_n$ terminent bien, alors $f(t_1, \ldots, t_n)$ termine bien. La preuve ressemble à celle de (B2). On ne détaille ici que le cas de la composition. Supposons que $t_1, \ldots, t_n$ terminent bien. On veut montrer que $(f \circ \langle g_1, \ldots, g_n \rangle)(t_1, \ldots, t_n)$ termine bien, ce qui se reformule en la bonne terminaison de $f(g_1(\vec{t}), \ldots, g_n(\vec{t}))$. Or par récurrence $g_1(\vec{t}), \ldots, g_n(\vec{t})$ terminent bien, et donc on peut conclure en appliquant à nouveau la récurrence. La deuxième étape consiste à montrer par récurrence sur la taille de t que t termine bien, pour tout $t \in T$, ce qui est évident au vu de la première étape.

(C2) Evident : la définition de $\to$ consiste essentiellement à orienter les égalités dénotationnelles.

(C3) On a par (C1), disons $t \to^\star S^m(\bot)$, puis $\underline{t} = \underline{S^m(\bot)} = S^m(\bot)$ par (C2).

(C4) Par récurrence sur la structure de f, et dans le cas $rec(g, h)(s, \vec{t})$ par récurrence sur $\text{coût}_S(s)$:

- $S(t)$. On a $\overline{S}(\underline{t}) = 1$, $\text{coût}_S(S(t)) = \text{coût}_S(t) + 1$ et $\text{coût}_{\to}(S(t)) = \text{coût}_{\to}(t)$, donc a fortiori $\text{coût}(t) \leq \text{coût}(S(t))$.
- $\pi_i^n(t_1, \ldots, t_n)$. On a $\overline{\pi_i^n}(\underline{t_1}, \ldots, \underline{t_n}) = i$, $\text{coût}_S(\pi_i^n(t_1, \ldots, t_n)) = \text{coût}_S(t_i)$ et $\text{coût}_{\to}(\pi_i^n(t_1, \ldots, t_n)) = \text{coût}_{\to}(t_i) + 1$, donc $\text{coût}(t_i) \leq \text{coût}(\pi_i^n(t_1, \ldots, t_n))$.

- $(f \circ \langle g_1, \ldots, g_n \rangle)(t_1, \ldots, t_m)$, avec $\overline{f}(\underline{g_1}(\underline{\vec{t}}), \ldots, \underline{g_n}(\underline{\vec{t}})) = j$. Par récurrence appliqueée à f, on a

$$\text{coût}(g_j(\vec{t})) \;\leq\; \text{coût}(f(g_1(\vec{t}), \ldots, g_n(\vec{t}))) \;<\; \text{coût}((f \circ \langle g_1, \ldots, g_n \rangle)(\vec{t})) \;.$$

  Puis, par récurrence appliqueée à g_j, on obtient $\text{coût}(t_i) \leq \text{coût}(g_j(\vec{t}))$, d'où la conclusion par transitivité.
- $rec(g,h)(s,\vec{t})$. Si $s \to^\star 0$, on note que $\text{coût}(g(\vec{t})) < \text{coût}(rec(g,h)(s,\vec{t}))$, et on conclut en appliquant la récurrence à g. Si $s \to^\star \perp$, alors

$$\text{coût}(s) = \text{coût}_\to(s) < \text{coût}_\to(rec(g,h)(s,\vec{t})) = \text{coût}(rec(g,h)(s,\vec{t})) \;.$$

  Si $s \to^\star S(u)$, soit $j = \overline{h}(\underline{u}, \underline{rec(g,h)}(\underline{u}, \underline{\vec{t}}), \underline{\vec{t}})$. On applique la récurrence à h. Si $j \geq 3$, on conclut d'une manière semblable au cas $s \to^\star 0$. Si $j = 1$, on obtient

$$\text{coût}(u) \;\leq\; \text{coût}(h(u, rec(g,h)(u,\vec{t}), \vec{t})) \;<\; \text{coût}(rec(g,h)(S(u),\vec{t})) \;.$$

  Si l est la longueur de la réduction $s \to^\star S(u)$, on a $\text{coût}(s) = \text{coût}(u) + l + 1$ et $\text{coût}(rec(g,h)(s,\vec{t})) = \text{coût}(rec(g,h)(S(u),\vec{t})) + l$, ce qui permet de conclure : $\text{coût}(s) < \text{coût}(rec(g,h)(s,\vec{t})) + 1$. Le cas $j = 2$ ressemble au cas que l'on vient de traiter et à celui de la composition (récurrence appliquée à $rec(g,h)(u,\vec{t})$). CQFD

(D1) Montrer, pour x de la forme $S^k(\perp)$, que si $\overline{h}(x, \underline{rec(g,h)}(x,\vec{y}), \vec{y}) = 2$, alors $\overline{rec(g,h)}(S(x),\vec{y}) = 1$.

(D2) Soit f p.r. d'arité n. Montrer que si $\underline{f}(\vec{x})$ est incomplet, alors il existe i, et $d_1, \ldots, d_n$ finis tels que $\vec{d} \leq \vec{x}$ et

$$\overline{f}(\vec{z}) = i \text{ pour tout } \vec{z} \text{ fini tel que } \vec{d} \leq \vec{z} \leq \vec{x} \;,$$

i.e., “f récurre ultimement sur le même argument”. (Cet énoncé n'a un contenu que si $\vec{x}$ a au moins une composante infinie.)

Preuve. (D1) On va montrer en fait (sous l'hypothèse de (D1))

$$\overline{rec(g,h)}(S(x),\vec{y}) = \overline{rec(g,h)}(x,\vec{y}) = \ldots = \overline{rec(g,h)}(\perp,\vec{y}) = 1$$

progressivement, à partir de la gauche, i.e., on montre par récurrence sur $l \leq k+1$ que si $\overline{h}(S^k(\perp), \underline{rec(g,h)}(S^k(\perp),\vec{y}), \vec{y}) = 2$, alors $\overline{rec(g,h)}(S^l(\perp),\vec{y}) = 1$. C'est vrai pour $l = 0$ par définition de $\overline{rec(g,h)}$. Pour le pas de récurrence, on remarque que les vecteurs

$$(S^l(\perp), \underline{rec(g,h)}(S^l(\perp),\vec{y}), \vec{y}) \text{ et } (S^k(\perp), \underline{rec(g,h)}(S^k(\perp),\vec{y}), \vec{y})$$

ne peuvent différer que sur leur deux premières composantes. Ceci implique que $\overline{h}(S^l(\perp), \underline{rec(g,h)}(S^l(\perp), \vec{y}), \vec{y}) \leq 2$, car sinon on aurait

$$\overline{h}(S^k(\perp), \underline{rec(g,h)}(S^k(\perp), \vec{y}), \vec{y}) = \overline{h}(S^l(\perp), \underline{rec(g,h)}(S^l(\perp), \vec{y}), \vec{y}) > 2$$

par (B3) : contradiction. Si $\overline{h}(S^l(\perp), \underline{rec(g,h)}(S^l(\perp), \vec{y}), \vec{y}) = 1$, on conclut directement, et si $\overline{h}(S^l(\perp), \underline{rec(g,h)}(S^l(\perp), \vec{y}), \vec{y}) = 2$, on conclut en utilisant l'hypothèse de récurrence.

(D2) Par récurrence structurelle. Les cas $f = S$ et $f = \pi_i^n$ sont immédiats, car les fonctions $\overline{S}$ et $\overline{\pi_i^n}$ sont constantes. Considérons $f \circ \langle g_1, \ldots, g_n \rangle$. Par récurrence, il existe j et $\vec{e}$ qui satisfont l'énoncé pour f et $(g_1(\vec{x}), \ldots, g_n(\vec{x}))$. Par (B4), $g_j(\vec{x})$ est incomplet. On peut donc appliquer la récurrence à g_j et à $\vec{x}$, et obtenir ainsi i et $\vec{d}$. On peut choisir $\vec{d}$ assez grand pour avoir $g_1(\vec{d}) \geq e_1, \ldots, g_n(\vec{d}) \geq e_n$ (par continuité, cf. (B2)). On a alors $\overline{f}(\underline{g_1}(\vec{z}), \ldots, \underline{g_n}(\vec{z})) = j$, d'où l'on déduit $(\overline{f \circ \langle g_1, \ldots, g_n \rangle})(\vec{z}) = \overline{g_j}(\vec{z}) = i$. Pour $rec(g,h)$, le cas $(\perp, \vec{y})$ ne s'applique pas. Le cas $(0, \vec{y})$ est facile (on applique la récurrence à g).
Reste le cas $(S(x), \vec{y})$. Comme $\underline{rec(g,h)}(S(x), \vec{y}) = \underline{h}(x, \underline{rec(g,h)}(x, \vec{y}), \vec{y})$ (et ce même si $x = S^\infty(\perp)$, par passage à la limite), on peut appliquer la récurrence à h : il existe j et $(d_0, e_0, \vec{d})$ tels que

$$(d_0, e_0, \vec{d}) \leq (z_0, t_0, \vec{z}) \leq (x, \underline{rec(g,h)}(x, \vec{y}), \vec{y}) \ \Rightarrow\ \overline{h}(z_0, t_0, \vec{z}) = j\ .$$

On peut choisir d_0 et $\vec{d}$ assez grands pour avoir $e_0 \leq \underline{rec(g,h)}(d_0, \vec{d})$ (par continuité). Si $j \geq 3$, on voit que $i = j - 1$ et $(S(d_0), \vec{d})$ conviennent. Semblablement, si $j = 1$, alors $i = 1$ et $(S(d_0), \vec{d})$ conviennent. Le cas $j = 2$ est plus complexe. On remarque d'abord que, par (B4), $\underline{rec(g,h)}(x, \vec{y})$ est incomplet. On distingue deux sous-cas :

- x est fini. Par récurrence, on peut trouver i, d'_0 et $\vec{d'}$ convenant à $rec(g,h)$ et à $(x, \vec{y})$. On peut choisir d'_0 et $\vec{d'}$ assez grands pour que $d'_0 = x$ et $\vec{d'} \geq \vec{d}$. Ce qui permet de combiner les bénéfices des deux hypothèses de récurrence. Soit $\vec{z}$ tel que $\vec{d'} \leq \vec{z} \leq \vec{y}$: alors $\overline{rec(g,h)}(S(x), \vec{z}) = \overline{rec(g,h)}(x, \vec{z}) = i$.
- $x = S^\infty(\perp)$. Ici (D1) offre une issue : si z_0, $\vec{z}$ sont tels que $d_0 \leq z_0 \leq x$ (donc z_0 est de la forme $S^m(\perp)$) et $\vec{d} \leq \vec{z} \leq \vec{y}$, on a $\overline{rec(g,h)}(z_0, \vec{z}) = 1$.

CQFD

La dernière question est une application à la complexité.

(E) Montrer qu'il n'existe pas de combinateur f calculant $\inf(m,n)$ en un temps $O(\inf(m,n))$. Plus précisément, montrer qu'il n'est pas possible d'avoir à la fois

$$\underline{f}(S^m(0), S^n(0)) = S^{\inf(m,n)}(0) \text{ et coût}_{\rightarrow}(f(S^m(0), S^n(0)) = O(\inf(n,p))\ .$$

PREUVE. On montre d'abord que s'il existe f satisfaisant les conditions de l'énoncé, alors $\underline{f}(S^\infty(\perp), S^\infty(\perp))$ est incomplet. Dans le cas contraire, on aurait, disons $\underline{f}(S^\infty(\perp), S^\infty(\perp)) = S^p(0)$; par continuité, il existerait m_0 et n_0 tels que

$$\underline{f}(S^m(\perp), S^n(\perp)) = S^p(0)$$

pour tous $m \geq m_0$, $n \geq n_0$. Mais ceci implique $\underline{f}(S^m(0), S^n(0)) = S^p(0)$ ($\underline{f}$ est croissante), ce que ne vérifie pas la fonction inf. On peut donc appliquer (D2) à f et $(S^\infty(\perp), S^\infty(\perp))$: il existe i, m_0 et n_0 tels que $\overline{f}(S^m(\perp), S^n(\perp)) = i$ pour tout $m \geq m_0$ et $n \geq n_0$. Disons que $i = 1$. Par (C4), ceci implique que $m = \text{coût}(S^m(\perp)) \leq \text{coût}(f(S^m(\perp), S^n(\perp))$. On peut montrer facilement que

$$\text{coût}_{\to}(f(S^m(\perp), S^n(\perp)) \leq \text{coût}_{\to}(f(S^m(0), S^n(0))$$

(plus généralement : $\text{coût}_{\to}(s) \leq \text{coût}_{\to}(t)$ si t est obtenu à partir de s en remplaçant certains $\perp$ par des termes arbitraires : ceci se montre par récurrence structurelle). Par ailleurs, par monotonie, on a aussi

$$\text{coût}_S(f(S^m(\perp), S^n(\perp)) \leq \text{coût}_S(f(S^m(0), S^n(0)) = \inf(m, n) \ .$$

On a donc $m \leq \text{coût}_{\to}(f(S^m(0), S^n(0)) + \inf(m, n)$. Ceci conduit à une contradiction, car on ne peut pas avoir $m \leq O(\inf(m, n))$. CQFD