

Option Informatique en Spé MP et MP*

DS du 22 mars 2004 : le corrigé

Langages rationnels et automates finis

Question 1 • Un mot u reconnu par \mathcal{A} au seuil 3 se décompose en $u = vw$ avec $1 \xrightarrow[v]{*} 2$ et $2 \xrightarrow[w]{*} 3$; donc $v = a^i b$, avec $0 \leq i \leq 2$ et $w = ba^j$ avec $0 \leq j \leq 2$. Donc $\mathcal{L}_3(\mathcal{A}) = \{bb, abb, bba, aabb, abba, bbaa, aabba, abbaa, aabbaa\}$.

Question 2 • Le plus court chemin menant de l'état initial à l'état final est de longueur 2; comme il n'y a aucune oasis, la longueur d'un calcul réussi est au moins égale à 2, et au plus égale à $k - 1$. Donc $\mathcal{L}_k(\mathcal{A}) = \emptyset$ pour $k \leq 2$. Supposons maintenant $k \geq 3$. Soit $p \in \llbracket 2, k-1 \rrbracket$; les mots de longueur p reconnus au seuil k sont de la forme $a^i b b a^j$ avec $i + j = p - 2$: il y en a $p - 1$. Donc $|\mathcal{L}_k(\mathcal{A})| = \sum_{2 \leq p < k} (p - 1) = \sum_{1 \leq p \leq k-2} p = \frac{(k-2)(k-1)}{2}$.

Remarquons que cette formule reste valable pour $k = 1$ et $k = 2$.

Question 3 • Il suffit de prendre $\Omega = Q$.

Question 4 • Les calculs réussis de \mathcal{A} sont de deux sortes :

- ceux qui ne passent par aucune oasis : leur longueur est au plus $k - 1$;
- ceux qui passent par au moins une oasis : notons $(q_j)_{0 \leq j \leq \ell}$ la suite des états par lesquels passe ce calcul, et j_1, \dots, j_q les indices (classés par ordre croissant) des oasis par lesquelles passe ce calcul. L'indice j_1 de la première oasis vérifie $0 \leq j_1 < k$; de même, l'indice j_q de la dernière oasis vérifie $\ell - k < j_q \leq \ell$. Les indices j_s et j_{s+1} de deux oasis consécutives vérifient $j_{s+1} - j_s \leq k + 1$.

Construisons un automate fini $\mathcal{A}' = (Q', \delta', I', F')$ qui reconnaît $\mathcal{L}_k(\mathcal{A})$:

- $Q' = Q \times \llbracket 0, k \rrbracket$;
- I' est la réunion de $I \times \{k - 1\}$ et $(I \cap \Omega) \times \{k\}$;
- $F' = F \times \llbracket 0, k \rrbracket$;
- la table de transition δ' est définie par les formules suivantes :
 - $((q, j), x(q', j - 1)) \in \delta'$ si $(q, x, q') \in \delta$ et $1 \leq j \leq k$;
 - $((q, j), x, (q', k)) \in \delta'$ si $(q, x, q') \in \delta$ et $q' \in \Omega$.

Dans l'état (q, j) de \mathcal{A}' , la deuxième composante j est l'autonomie de l'automate (le nombre de transitions qu'il peut effectuer sans passer par une oasis). En rendant $(i, k - 1)$ initial, nous autorisons les calculs du premier type, et les calculs menant d'un état initial à une première oasis.

Question 5 • Si q n'est pas une oasis, l'état (q, k) n'est pas accessible : nous pouvons donc l'éliminer. Si q' est une oasis, la transition $((q, j), x(q', j - 1))$ est inutile. Nous pouvons donc éliminer les états de la forme (q', j) $q' \in \Omega$ et $j < k$. Finalement, nous ne gardons que les états de la forme (q, j) avec $q \notin \Omega$ et $0 \leq j < k$; et ceux de la forme (q, k) avec $q \in \Omega$. Nous n'avons donc besoin que de $k(|Q| - |\Omega|) + |\Omega| = kn - (k - 1)|\Omega|$ états.

Algorithmique

Question 1 • Après la première action, les deux mémoires contiennent la même valeur a .

Question 2 • Notons a_k (resp. b_k) le contenu de la mémoire a (resp. b) après la k -ième action. Nous avons $a_1 = a_0 + b_0$ et $b_1 = b_0$; $a_2 = a_1 = a_0 + b_0$ et $b_2 = a_1 - b_1 = a_0$; $a_3 = a_2 - b_2 = b_0$ et $b_3 = b_2 = a_0$. Conclusion : la suite d'actions proposée échange les contenus des mémoires a et b .

Question 3 • $a \leftarrow b - a$; $a \leftarrow a - b$.

Question 4 • La CNS s'énonce ainsi: a et b sont premiers entre eux. Elle est nécessaire car, notant d le PGCD de a et b , chaque action envisageable remplace a ou b par un multiple de d . Elle est suffisante: nous nous ramenons au cas où a et b sont positifs; puis, tant que a et b sont distincts, nous retranchons le plus petit du plus grand. Les valeurs de a et b restent premières entre elles; leurs somme décroît strictement, donc au bout d'un nombre fini d'étapes, $a = b = 1$.

Question 5 • Le cycle $(1, 2, 3)$ est le produit des deux transpositions $\tau_{1,2}$ et $\tau_{2,3}$. En utilisant le résultat de la question 2, nous construisons une suite de six actions répondant à la question. Il serait intéressant de chercher s'il existe une suite de cinq actions réalisant cette permutation circulaire. Indication: parcours en profondeur dans un graphe, en limitant la profondeur à cinq.

Question 6 • $\text{let } (a,b,c) = (b,c,a)$.

Algorithmique des intervalles de \mathbb{Z}

Question 1 • Il suffit de remarquer qu'une partie de \mathbb{Z} est représentable ssi elle est finie.

Question 2 • Soit \mathcal{G} une représentation de A de cardinal minimal. Soient \mathcal{I} et \mathcal{J} deux intervalles appartenant à \mathcal{G} ; la réunion de \mathcal{I} et \mathcal{J} n'est pas un intervalle, donc nous avons nécessairement $\mathcal{I} \prec \mathcal{J}$ ou $\mathcal{J} \prec \mathcal{I}$.

Question 3 • Soit \mathcal{G} une représentation de A : chaque intervalle appartenant à \mathcal{G} contient au moins un élément de A , donc $\mathcal{C}(A) \leq |A|$. Soit $A = \{2, 4, \dots, 2n\}$: $|A| = \mathcal{C}(A) = n$.

Question 4 • Commençons par trier les éléments de \mathcal{F} par extrémité gauche croissante, pour un coût $\mathcal{O}(n \ln(n))$; notons $\mathcal{F}' = (\mathcal{I}_j)_{1 \leq j \leq n}$ la liste obtenue. Notons que $\mathcal{I}_j \prec \mathcal{I}_{j+1}$ implique $\mathcal{I}_j \prec \mathcal{I}_k$ pour tout $k > j$. Il suffit donc de balayer la liste \mathcal{F}' en fusionnant deux éléments consécutifs chaque fois que c'est nécessaire. Plus précisément, si la liste contient moins de deux intervalles, c'est fini; sinon, notons $\llbracket a, b \rrbracket$ le premier intervalle et $\llbracket c, d \rrbracket$ le deuxième. Alors:

- si $b < c - 1$, garder le premier intervalle et continuer le balayage;
- sinon, les remplacer par $\llbracket a, d \rrbracket$

Question 5 • Aucune difficulté: nous faisons glisser l'intervalle \mathcal{J} à insérer le long de la liste, pour lui faire dépasser tous ceux qui le précèdent selon l'ordre \prec . Si nous nous arrêtons en fin de liste, ou avant un intervalle \mathcal{K} tel que $\mathcal{J} \prec \mathcal{K}$, il suffit de placer \mathcal{J} à cet endroit; sinon, il faut fusionner \mathcal{J} et \mathcal{K} et poursuivre le balayage avec l'intervalle \mathcal{J}' résultant de la fusion.

```
let rec ajoute (a,b) = fonction
  | [] -> [(a,b)]
  | (g,d)::q when b < g-1 -> (a,b)::(g,d)::q
  | (g,d)::q when a > d+1 -> (g,d)::(ajoute (a,b) q)
  | (g,d)::q -> ajoute (min a g, max b d) q ;;
```

Question 6 • $A - \mathcal{J}$ est finie.

Question 7 • Il faut examiner différents cas, selon la place de l'intervalle $\llbracket a, b \rrbracket$ par rapport à la tête de liste.

```
let rec retranche_intervalle (a,b) = fonction
  | [] -> []
  | (g,d)::q when b < g -> (g,d)::q
  | (g,d)::q when d < a -> (g,d)::(retranche_intervalle (a,b) q)
  | (g,d)::q when a <= g & d <= b -> retranche_intervalle (a,b) q
  | (g,d)::q when a <= g & b < d -> (b+1,d)::q
  | (g,d)::q when g < a & d <= b -> (g,a-1)::(retranche_intervalle (a,b) q)
  | (g,d)::q -> (g,a-1)::(b+1,d)::q ;;
```

Question 8 • A et B ne sont pas vides, donc $A \cup B$ ne l'est pas; ainsi, $\mathcal{C}(A \cup B) \geq 1$. D'autre part, $\mathcal{C}(A \cup B) \leq |A \cup B| \leq |A| + |B| = p + q$.

Question 9 • Supposons $p \geq q$ pour fixer les idées. Prenons $A = \{2, 4, \dots, 2p\}$ et $B = \{1, 3, \dots, 2q - 3\} \cup [2q - 1, 2p - 1]$. Alors $A \cup B = \llbracket 1, 2p \rrbracket$ donc $\mathcal{C}(A \cup B) = 1$.

Question 10 • Prenons $A = \{2, 4, \dots, 2p\}$ et $B = \{-2, -4, \dots, -2q\}$: alors $\mathcal{C}(A \cup B) = p + q$.

Question 11 • $\mathcal{C}(A \cap B) \geq 0$, avec égalité ssi A et B disjointes. Je dis que $\mathcal{C}(A \cap B) \leq p + q - 1$, l'égalité étant possible.

- Raisonçons par récurrence sur $p + q$. Pour $p + q = 2$, le résultat est clair puisque A et B sont des intervalles. Supposons le résultat acquis lorsque $p + q = n$, et soient A et B deux parties représentables, de complexités respectives p et q avec $p + q = n + 1$. Notons \mathcal{I} le dernier intervalle de la représentation canonique de B , et $B' = B - \mathcal{I}$. Notons s le nombre d'intervalles de la représentation canonique de A qui ont une intersection non vide avec B' ; alors la complexité de $A \cap B'$ est au plus $s + q - 1$. L'intersection de \mathcal{I} et A est l'intersection de \mathcal{I} et des $p - s + 1$ derniers intervalles de la représentation canonique de A , donc la complexité de $A \cap \mathcal{I}$ est au plus $p - s + 1$. Finalement, la complexité de $A \cap B$ est au plus $p + q = n$, ce qui termine la preuve.

- Il reste à exhiber un couple (A, B) pour lequel on a l'égalité. Supposons $p \leq q$ pour fixer les idées. Nous prendrons pour A la réunion des intervalles $\llbracket 4k, 4k + 2 \rrbracket$, pour $1 \leq k \leq p$; et B sera la réunion des intervalles $\llbracket 4k + 2, 4k + 4 \rrbracket$ pour $1 \leq k < q$ et (au besoin) de l'intervalle $\llbracket 4q + 2, 4p + 2 \rrbracket$. Alors $A \cap B$ sera la réunion de l'ensemble des naturels pairs de 6 à $4p$ inclus, et des intervalles $\llbracket 4k, 4k + 2 \rrbracket$, pour $q + 1 \leq k \leq p$. Ce qui nous fait bien une complexité de $p + q - 1$.

Question 12 • Très peu de différences avec la recherche dans un arbre binaire de recherche habituel.

```
let rec cherche x = fonction
  | F -> false
  | N(g,_,tg,_) when x < g -> cherche x tg
  | N(_,d,_,td) when x > d -> cherche x td
  | N(_,_,_,_) -> true ;;
```

Question 13 • Parcours en profondeur de l'arbre; l'intervalle discret $\llbracket p, q \rrbracket$ contient $q - p + 1$ éléments.

```
let rec taille = fonction
  | F -> 0
  | N(g,d,tg,td) -> taille(tg) + taille(td) + d - g + 1 ;;
```

Question 14 • Simple adaptation de la recherche du plus petit élément dans un arbre binaire de recherche.

```
let rec minimum = fonction
  | F -> failwith "arbre vide"
  | N(g,_,F,_) -> g
  | N(_,d,tg,_) -> minimum tg ;;
```

Question 15 • Avec intervalle et la concaténation de listes, c'est facile :

```
let rec énumère = fonction
  | F -> []
  | N(g,d,tg,td) -> (énumère tg) @ (intervalle g d) @ (énumère td) ;;
```

Question 16 • Le coût peut être quadratique en la taille de l'arbre.

Question 17 • L'insertion de x dans un arbre d'intervalles contenant les intervalles $\llbracket x - 1, x - 1 \rrbracket$ et $\llbracket x + 1, x + 1 \rrbracket$ ne se fera pas correctement.

Question 18 • Si $\mathcal{I} \prec \mathcal{R}$ ou $\mathcal{R} \prec \mathcal{I}$ ou $\mathcal{I} \subset \mathcal{R}$, la racine n'est pas modifiée. Sinon, \mathcal{R} et \mathcal{I} vont être fusionnés; si cette fusion diminue (resp. augmente) la borne inférieure (resp. supérieure) de \mathcal{R} , il faudra parcourir le sous-arbre gauche (resp. droit) pour réaliser au besoin d'autres fusions. Ceci se produit dans cinq situations :

- \mathcal{I} et \mathcal{R} sont consécutifs;
- \mathcal{R} et \mathcal{I} sont consécutifs;
- \mathcal{I} et \mathcal{R} ne sont pas disjoints, et aucun n'est contenu dans l'autre (deux cas);
- $\mathcal{R} \subset \mathcal{I}$.

Question 19 • Pas de difficulté particulière :

```

let rec étend_g a = function
| F -> (a,F)
| N(g,d,tg,td) when a<g -> étend_g a tg
| N(g,d,tg,td) when a<=d -> (g,tg)
| N(g,d,tg,td) -> let (a',td') = étend_g a td
  in (a',N(g,d,tg,td')) ;;

```

Question 20 •

```

let rec ajoute_intervalle (a,b) = function
| F -> N(a,b,F,F)
| N(g,d,tg,td) as t when g<=a & b<=d -> t
| N(g,d,tg,td) when b<g -> let tg' = ajoute_intervalle (a,b) tg
  in N(g,d,tg',td)
| N(g,d,tg,td) when d<a -> let td' = ajoute_intervalle (a,b) td
  in N(g,d,tg,td')
| N(g,d,tg,td) when a<g & b<=d -> let (g',tg') = étend_g a tg
  in N(g',d,tg',td)
| N(g,d,tg,td) when g<=a & d<b -> let (d',td') = étend_d b td
  in N(g,d',tg,td')
| N(g,d,tg,td) -> let (g',tg') = étend_g a tg
  and (d',td') = étend_d b td in N(g',d',tg',td') ;;

```

Question 21 • Insérer x revient à ajouter l'intervalle discret $\llbracket x, x \rrbracket$.

```

let insertion x = ajoute_intervalle (x,x) ;;

```

Question 22 • Adaptation de l'algorithme de scission d'un arbre binaire de recherche; si l'on trouve un intervalle contenant le point de coupure x , il y a quatre cas à examiner selon que x est ou n'est pas la borne gauche et/ou la borne droite de cet intervalle.

```

let rec scission x = function
| F -> (F,F)
| N(g,d,tg,td) when x<g -> let (tgg,tgd) = scission x tg
  in (tgg,N(g,d,tgd,td))
| N(g,d,tg,td) when x>d -> let (tdg,tdd) = scission x td
  in (N(g,d,tg,tdg),tdd)
| N(g,d,tg,td) when x=g & g=d -> (tg,td)
| N(g,d,tg,td) when x=g -> (tg,N(g+1,d,F,td))
| N(g,d,tg,td) when x=d -> (N(g,d-1,tg,F),td)
| N(g,d,tg,td) -> (N(g,x-1,tg,F),N(x+1,d,F,td)) ;;

```

Le coût de la scission est un \mathcal{O} de la hauteur de l'arbre t . Signalons une solution brutale, qui consiste à construire la liste ordonnée des éléments décrits par t , à scinder cette liste, et à reconstruire les arbres d'intervalles associés aux deux listes :

```

let scinde_liste x l =
  let rec aux accu = function
  | [] -> (accu,[])
  | t::q when t<x -> aux (t::accu) q
  | t::q when t=x -> (accu,q)
  | q -> (accu,q)
  in aux [] l ;;

let interv_tree_of_list l = list_it insertion l F ;;

let scission_brutale x t =
  let (lg,ld) = scinde_liste x (énumère t)
  in (interv_tree_of_list lg,interv_tree_of_list ld) ;;

```

FIN