Option informatique

Présentation du sujet

Le sujet portait sur des algorithmes classiques liant automates et expressions rationnelles. Il se décomposait en trois parties indépendantes :

– La première partie, qui représentait la moitié de l'épreuve, abordait des questions autour du miroir d'un langage et autour de la rationalité de certains langages construits avec l'opérateur miroir, puis dans un deuxième temps, implémentait l'algorithme de déterminisation d'un automate pour construire au final l'automate déterministe minimal via l'algorithme de Brzozowski.

– Après avoir introduit un type `OCaml` pour les expressions rationnelles, la deuxième partie faisait manipuler cette implémentation à travers diverses questions puis proposait de programmer l'algorithme dichotomique de Conway qui permet, à partir du calcul de l'étoile d'une matrice d'expressions rationnelles, de calculer une expression rationnelle du langage d'un automate.

– La troisième partie introduisait les dérivées d'expressions rationnelles d'Antimirov, construisait un automate associé à une expression rationnelle E, et démontrait qu'il reconnait bien le langage associé à E et que son nombre d'états est relativement réduit.

Les chapitres abordés étaient : langage et automates, algorithmes diviser pour régner et complexité.

Analyse globale des résultats

Le sujet était relativement long et la troisième partie, plus courte mais plus difficile, n'a été abordée sérieusement que dans les meilleures copies. Néanmoins, quelques rares candidats ont réussi à traiter la quasi-totalité de l'épreuve et chaque question a été traitée de manière satisfaisante par au moins une copie.

L'épreuve comportait, à parts à peu près égales, des questions de programmation et des questions théoriques sur les langages et les automates. La partie programmation est dans l'ensemble plutôt réussie avec une bonne maitrise des bases du langage Ocaml sur la plupart des copies même si l'on rencontre encore trop souvent des confusions avec Python sur la manipulation des listes ou les bornes dans les boucles `for`. Le sujet permettait d'utiliser librement toute fonction des modules `List` ou `Array`. Si l'usage de fonctions élaborées du type `List.fold_left` ou `List.iter` ne fait évidemment pas partie des attendus voire est déconseillé car cela produit des codes vite obscurs et parfois non optimaux, les candidats ne doivent pas hésiter à utiliser les fonctions les plus basiques (du moment qu'ils en connaissent les complexités) comme par exemple `List.length`, `List.mem` ou `Array.make_matrix`. Trop d'étudiants perdent du temps à les reprogrammer, parfois maladroitement.

Les réponses apportées aux questions théoriques (tant sur les automates que dans les calculs de complexité) ont été par contre souvent plus confuses ou sans véritable justification, ou s'appuyaient sur des résultats hors programme pas toujours bien maitrisés (lemme de pompage, théorème donnant directement la solution d'une récurrence $C(n) = aC(\frac{n}{2}) + O(n^b)$).

Dans la gestion du temps, certains candidats se sont obstinés sur la première partie en oubliant que les parties étaient indépendantes et se sont ainsi privés des points largement abordables de la partie II.

Enfin, le jury a noté cette année une amélioration certaine dans la présentation des copies mais observe encore trop de fautes d'orthographe grossières et encourage les candidats à faire un effort sur ce point.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Citons d'abord les erreurs ou maladresses les plus fréquemment rencontrées dans les codes proposés :

— l'oubli quasi-systématique des parenthèses dans le passage des paramètres à une fonction (par exemple `f\ n - 1` au lieu de `f\ (n-1)`) ou de délimiteurs type `begin/end`. L'indentation, certes utile pour comprendre le code, n'est pas un délimiteur comme en Python. Dans le code

```
if n>0 then
    a:= !a + 1 ;
    f (n-1) ;
```

la fonction `f` sera appelée que `n` soit strictement positif ou pas.

— l'oubli du mot clé `ref` ou du `!` dans la gestion des variables. Si un oubli isolé est souvent pardonné, à l'inverse, leur absence systématique est clairement sanctionnée.

— la confusion entre le caractère `'a'` de type `char` et une variable nommée `a`.

— des formulations alambiquées comme par exemple `let x= aux n in x` (simplifiable en `aux n`) ou bien `if test = true then true else\ false` (simplifiable en `test`)

— la définition de fonctions auxiliaires à l'intérieur d'une fonction récursive. Mieux vaut définir cette fonction indépendamment.

— l'usage systématique de fonctions récursives : si programmation récursive et programmation itérative sont interchangeables, certaines fonctions, de part les structures informatiques qu'elles manipulent (des tableaux par exemple), sont plus naturelles à écrire avec des boucles for ou while (**Q6**, **Q36**). Dans le même ordre d'idée, certains candidats cherchent à tout prix à écrire des fonctions récursives terminales à l'aide d'accumulateurs là où des fonctions récursives « ordinaires » font aussi bien l'affaire (et sont plus simples à écrire et à comprendre).

On précise qu'à de très rares exceptions près, la plupart des programmes s'écrivaient en peu de lignes et avec une fonction auxiliaire au maximum. La programmation de **plusieurs** fonctions intermédiaires pour répondre à une question donnée ou l'usage de conversion de types (tableau $\leftrightarrow$ liste) devrait être un signe qui alerte le candidat sur une méthode maladroite ou un algorithme à la complexité dégradée (ce qui sera pénalisé, en particulier si aucune explication n'est fournie pour expliquer la démarche choisie). Par exemple, écrire en **Q22** un sous-programme qui parcourt n^2 fois la liste des transitions de l'automate est à coup sûr sous-optimal (même s'il est stipulé que $n \leqslant 20$!).

C'est un lieu commun mais il est important de bien lire chaque (sous-)partie dans sa totalité afin de bien en comprendre l'esprit. Par exemple, dans la partie I.C visant à programmer l'algorithme de déterminisation, on faisait prendre conscience au candidat que manipuler des ensembles d'états sous forme de listes d'états menait souvent à des fonctions auxiliaires de complexité un peu élevée. On expliquait alors qu'on allait prendre une autre implémentation en s'aidant de la représentation binaire des entiers, ce qui permet, tant que le nombre d'états n de l'automate original ne dépasse pas le nombre de bits autorisés pour la représentation des entiers OCaml, d'avoir en Q19 une fonction d'appartenance à un ensemble en $O(1)$ tout en restant raisonnable dans l'utilisation de l'espace mémoire (contrairement par exemple à la fonction `List.mem` qui est en $O(n)$). De nombreux candidats n'ont pas compris la démarche et ont manipulé ou généré dans leurs programmes des ensembles d'états sous forme de listes avant de les convertir à la fin sous la forme attendue au lieu de travailler directement avec la représentation choisie (ce qui a pour effet d'annuler l'optimisation proposée par le sujet) (**Q20**, **Q22**, **Q24**).

Sur le plan théorique, le jury a rencontré beaucoup de confusions dans les objets manipulés :

- confusion entre a^*b^* qui est un langage (ici l'ensemble des mots $\{a^n b^p,\ (n,p) \in \mathbb{N}^2\}$) et $\{a^*b^*\}$ qui est un ensemble de langages (**Q1**, **Q9**).
- confusion entre $\{L_{q,q'},\ (q,q') \in I \times F\}$ et $\bigcup_{(q,q')\in I\times F} L_{q,q'}$ (**Q9**)
- confusion entre $(\Sigma^2)^*$ (ensemble des mots de longueur paire) et $(\Sigma^*)^2$ ensemble des concaténations de deux mots (qui vaut Σ^* donc)

On rappelle qu'un sous-langage d'un langage rationnel n'est pas forcément rationnel (sinon, comme Σ^* est rationnel, tout langage serait rationnel).

Dans les questions de construction d'automates, il n'est pas permis d'étiqueter des transitions par des mots (ε ou ab par exemple) et quand on propose un automate qui reconnait un langage donné, il faut s'assurer qu'il reconnait bien tous les mots du langage mais seulement eux (**Q13**). Lors de la déterminisation (**Q14**, **Q15**), il convient de s'assurer que l'automate dessiné à la fin est bien déterministe et qu'on a bien porté les états finaux.

Dans les questions **Q26** et **Q27**, il était préférable, afin de fournir des preuves claires, de revenir à des choses simples en considérant des chemins dans l'automate $\mathcal{A}$. On a noté beaucoup de confusions entre accessible (ce qu'était par définition l'automate $\mathcal{A}_{det}$) et coaccessible (ce qu'était l'automate $\mathcal{A}$ car son miroir était supposé accessible).

Dans la partie II.A, signalons que le langage vide n'est pas seulement représenté par l'expression $\emptyset$ (cf $\emptyset.(\varepsilon + a)$). D'autre part, une expression sans lettre ne correspond pas forcément au langage vide (ε par exemple). La question **Q34** est souvent mal traitée : pour simplifier par exemple l'expression donnée en **Q33**, il faut agir récursivement en profondeur. Trop de candidats se limitent à tester s'il y a une simplification possible à la racine, produisant même parfois des boucles infinies.

Les questions **Q35** et **Q36** étaient assez abordables mais de nombreux candidats prennent la taille des matrices comme des variables globales (l'énoncé ne disait rien de tel) ou ne semblent pas connaitre `Array.make_matrix` (`Array.make n Array.make p Vide`) ne crée pas une vraie matrice dont les lignes sont indépendantes). De plus, la complexité demandée est parfois manquante. On ne saurait trop conseiller aux candidats, une fois chaque question terminée, de relire l'énoncé de ladite question pour voir s'il n'en a pas oublié une partie.

Les questions **Q38** et **Q39** ont causé pas mal de difficultés. Au moment de compter les appels récursifs, certains candidats oublient que des calculs intervenant dans plusieurs formules à la fois peuvent être mémorisés pour ne pas avoir à être faits plusieurs fois (erreur que l'on retrouve en **Q41**. La résolution des relations de récurrence est très rarement bien faite. On rappelle qu'une technique simple pour une relation $C(n) = aC(n-1) + O(f(n))$, consiste à introduire $D(n) = \frac{C(n)}{a^n}$ ce qui ramène à l'estimation de la somme $\sum_{k=1}^{n} \frac{f(k)}{a^k}$.

Pour le reste du sujet, hormis la question **Q44**, abordée par un nombre conséquent de copies et qui a donné lieu à beaucoup d'erreurs de calculs, les questions de la partie II.C et de la partie III, quand elles ont été abordées sérieusement, ont été, dans l'ensemble, plutôt bien traitées.

Conclusion

Le jury a conscience que le sujet portait sur une partie difficile du programme (langages et automates) et encourage les futurs candidats à bien maitriser leur cours sur ces chapitres.

De façon générale, afin de préparer au mieux l'épreuve d'option informatique, Il convient de s'imposer un entrainement régulier sur machine, seul moyen d'acquérir de bons réflexes en programmation et de s'habituer à rédiger en détail des questions théoriques afin de gagner en aisance dans les argumentations.