4.1.3 Conseils aux futurs candidats

Nous conseillons aux futurs candidats une lecture attentive du rapport du jury afin d'éviter les erreurs récurrentes dans les copies.

Nous conseillons de se familiariser avec le programme officiel d'informatique commune afin de vérifier la maîtrise des points exigibles aux concours.

4.2 Informatique option MP

4.2.1 Généralités

Le sujet de cette année porte sur les langages rationnels et propose de construire un automate reconnaissant un langage L donné à partir des réponses fournies par un oracle qui sait répondre aux questions

- un mot w appartient-il au langage L ?
- un automate $\mathcal{A}$ reconnaît-il le langage L et, si la réponse est non, donner un contre-exemple, c'est-à-dire un mot appartenant à L mais non reconnu par $\mathcal{A}$ ou un mot reconnu par $\mathcal{A}$ mais n'appartenant pas à L.

Partie 1. Cette partie permet de s'approprier les premières notions du sujet avec quelques questions de cours et plusieurs fonctions simples à écrire en `OCaml`, certaines donnent un cadre pour guider l'écriture du code.

Partie 2. Le sujet introduit la notion de séparabilité de deux mots par un mot et fait travailler sur cette notion et la relation d'équivalence associée.

Partie 3. Le sujet utilise la séparation de mots dans un arbre de décision et étudie des propriétés de ceux-ci.

Partie 4. Cette partie donne la définition d'un automate à partir de certains arbres par itération de constructions. La dernière question montre qu'on a construit un automate minimal.

On remarque que l'oracle peut être défini à partir d'un automate reconnaissant un langage L : la construction permet donc de minimiser un automate.

12 des 28 questions demandent d'écrire des fonctions dans le langage `OCaml`.

Une analyse détaillée des questions est présentée dans l'annexe R.

4.2.2 Commentaires généraux

- Les correcteurs ont été surpris par le nombre de candidats, de l'ordre du tiers des copies, qui écrivent

  ```
  if condition then true else false
  ```

  Bien entendu cette construction donne le bon résultat mais elle laisse entendre un manque de compréhension des booléens.

- L'usage d'une fonction récursive auxiliaire est une bonne pratique quand on a besoin de faire intervenir des variables supplémentaires mais cela devient inutile et même contre-productif quand la fonction auxiliaire reprend exactement les variables de la fonction.

```
let f a b c =
  let rec aux a b c =
    ...
  in
aux a b c
```

  On a le droit d'écrire directement une fonction récursive.

- De même un raisonnement par l'absurde permet d'écrire des preuves plus simplement. Cependant ce type de raisonnement obscurcit les choses quand il ne sert que d'enveloppe à un raisonnement direct. Une part non négligeable de candidats, pour prouver par exemple une égalité $a = b$, écrivent

  Je suppose qu'on a $a \neq b$. *Or je prouve que* $a = b$ *donc j'aboutis à une contradiction. J'en déduis que* $a = b$.

On a une sorte de mise en abyme qui n'aide pas à la compréhension.

4.2.3 Conseils aux futurs candidats

- Les réponses aux questions de programmation montre un manque de solidité dans l'écriture des codes chez certains candidats. Il est utile aussi de continuer le codage en dehors des heures de cours afin de maîtriser le langage.

- Quand on écrit des fonctions dans le langage de programmation, il est indispensable qu'elles soient compréhensibles : donnez des noms explicites aux variables et aux fonctions auxiliaires, expliquez à quoi correspondent les variables, expliquez ce que fait une fonction auxiliaire, ne surchargez pas les lignes par des ratures et des ajouts, écrivez une fonction sur une seule page et de manière linéaire sans renvoi fléché vers un bout de code, ...
  Un bon test est de se relire après quelques minutes : si vous ne comprenez pas ce que vous avez fait, comment le correcteur pourrait-il comprendre ?

- Les questions de dénombrements ne sont pas faciles même (surtout ?) lorsque le résultat semble évident. Vous ne perdrez pas votre temps en écrivant une description précise des ensembles et des fonctions bijectives, injectives ou surjectives que vous utilisez.

- Comme dans toutes les épreuves, il est indispensable de bien lire l'énoncé. Bien des réponses infructueuses sont le fruit d'une lecture défaillante. Lisez l'énoncé, re-lisez le.

- Soignez votre lisibilité, il ne suffit pas de comprendre, il faut aussi faire comprendre que vous avez compris. Évaluer une copie dont on doit décrypter les lignes qui ressemblent à des vaguelettes est impossible.

4.2.4 Conclusions

Même si les remarques et conseils ci-dessus ont mis l'accent sur les problèmes que les correcteurs ont rencontrés, il faut souligner l'investissement de la plupart des candidats. Ils se sont appropriés l'énoncé avec ses notations et ses concepts nouveaux, ils ont avancé dans les questions et apporté des réponses souvent pertinentes. La compréhension de la matière est visible chez la majorité des candidats.

4.3 Informatique 1 filière MPI

4.3.1 Généralités et présentation du sujet

Le sujet s'intéresse à différentes façons d'implémenter en langage C les tableaux associatifs. Il est composé de quatre parties indépendantes comprenant au total 34 questions.

La première partie s'intéresse aux arbres binaires de recherche équilibrés. Les questions de programmation sont pour la plupart des questions proches du cours, et ont été globalement réussies. Les quelques questions théoriques n'ont pas eu une grande réussite.

La seconde partie s'intéresse à l'implémentation d'un ramasse-miettes. Elle ne comporte quasiment que des questions de programmation. Cette partie a été globalement réussie.

La troisième partie traite de la structure de skip list. La moitié des questions sont des questions de programmation, l'autre moitié des questions théoriques. Cette partie, plus difficile que les autres, a mis en difficulté beaucoup de candidats.

La dernière partie concerne l'analyse syntaxique d'une chaîne de caractères encodant un tableau associatif. Elle ne comportait que des questions théoriques, liées aux grammaires et les automates. Les candidats qui ont abordé cette partie l'ont fait souvent avec succès.

Une analyse détaillée des questions est présentée dans l'annexe S.

4.3.2 Commentaires généraux

- Les codes attendus sont pour la plupart courts. Écrire des fonctions dépassant la dizaine de lignes augmente le risque d'écrire du code incorrect et doit être perçu comme un signe de complexité excessive.
- En C, il ne faut pas imbriquer des fonctions les unes dans les autres « à la OCaml », cela n'est pas correct.
- Lorsqu'une fonction doit renvoyer une référence vers une structure qui doit être allouée par la fonction, il faut allouer la mémoire dans le tas (en utilisant la fonction `malloc`) et non dans la pile (en déclarant une variable locale de type structure).
- Lorsqu'une variable est de type référence, il n'est pas forcément nécessaire de faire une allocation mémoire, cela peut simplement être un passage de paramètre.
- Une attention toute particulière a été apportée à la validité du code C présenté.
- Numéroter les questions est essentiel. Numéroter les sections n'est pas nécessaire.
- Mettre en valeur le code ou les résultats principaux est apprécié par le correcteur.
- Certains codes sont très difficiles à comprendre parce qu'ils contiennent trop de ratures ou de corrections.

R Informatique option MP

Q1 - Une question de programmation simple mais pour laquelle l'erreur d'utiliser une variable comme motif a parfois été commise. **Q2 -** L'ordre lexicographique ne semble pas être connu par une partie des candidats. **Q3 -** La réponse à cette question ne saurait être de proposer l'utilisation du module `Hashtbl` : ce n'est pas une structure de données. De plus les fonctions du module indiquent que la structure de donnée utilisée est mutable, ce qui ne répond pas à la question de l'énoncé. **Q4 -** Il s'agissait ici d'utiliser les fonctions définies dans l'énoncé et non de donner la description de dictionnaires par des ensembles de couples. **Q5 -** Pour cette question certains ont bien vu le langage reconnu mais ont donné des expressions régulières qui ne le dénotaient pas. D'autres ont donné une bonne expression régulière, sans doute en appliquant un algorithme du cours, mais n'ont pas su donner une description du langage reconnu. **Q6 -** Beaucoup de candidats ne semblent pas savoir ce qu'est un automate de Glushkov. **Q7 -** Peu de candidats ont lu dans l'énoncé que `final` devait être une fonction. **Q8 -** Question souvent bien traitée. **Q9 -** Question souvent bien traitée. **Q10 -** Une réponse qui fait que `separated_by u v w` renvoie `false` lorsque u et v sont séparés par w ne saurait être correcte. **Q11 -** Cette question semble avoir désarçonné une partie des candidats, peut-être parce qu'elle est "trop simple". **Q12 -** Dans cette question, il fallait justifier que, si deux mots parviennent à un même état, alors, quel que soit le suffixe qu'on leur ajoute à ces mots, les transitions associées arrivent toujours dans un même état. Parfois ce fait est juste énoncé au milieu de raisonnements inutiles, la question n'apporte alors pas de points. **Q13 -** Demander de citer un théorème est sans doute une idée folle : il y a eu de nombreux noms proposés, parfois proches (Klein, Quine, Klein-Gordon) parfois moins (lemme de l'étoile). Donner le nom ne suffit bien entendu pas : *Le théorème de Kleene relie les langages reconnaissables et les langages rationnels* n'est pas une réponse satisfaisante. La démonstration de la finitude est trop souvent brouillonne. Cela se retrouve dans les questions 20. et 28. **Q14 -** Cette question demande une complexité linéaire. Beaucoup de codes proposés ne respectent pas cette complexité tout en fournissant une "preuve" de la linéarité. **Q15 -** Question facile mais quelques candidats n'ont pas compris le criblage. **Q16 -** Question souvent réussie. **Q17 -** Pour répondre simplement il fallait supposer que la fonction μ était croissante, ce qui ne semble pas avoir posé de problème. Quelques candidats calculent la complexité d'une succession d'instructions avec un produit. **Q18 -** Dans cette question il y a eu parfois des raisonnements compliqués et faux. **Q19 -** C'est la contraposée de la question précédente. **Q20 -** Une partie des candidats prouve en fait que le nombre de feuilles accessibles est borné : il est nécessaire de rappeler que toutes les feuilles sont accessibles. **Q21 -** Question souvent bien traitée. **Q22 -** Même remarque qu'à la question 7. pour `finals`. **Q23 -** L'arbre doit être un crible : ce n'est pas souvent respecté. La contrainte du nombre de feuilles n'a pas toujours été comprise. **Q24 -** Très peu de candidats ont su donner une preuve correcte. **Q25 -** Beaucoup de candidats ont répondu à la question en essayant d'écrire des fonctions de découpages de listes. C'est une méthode pour arriver au résultat mais, trop souvent, le manque d'explications du code rend celui-ci illisible. **Q26 -** Très peu de candidats ont vu qu'il y avait 2 cas de construction dans le cas d'une feuille et 2 cas d'appel récursif dans un seul des deux fils d'un nœud. **Q27 -** Cette question rassemblait de nombreux résultats antérieurs : quelques candidats ont réussi à les ordonner correctement. **Q28 -** Dernière question peu traitée.

↑RETOUR