Option Informatique

Présentation du sujet

Le sujet 2019 de l'option informatique s'intéresse à l'étude de l'énumération de Gray des écritures binaires et leur utilisation pour générer efficacement des combinaisons dans un ensemble à n éléments. La première partie traite de l'énumération lexicographique puis dans l'ordre de Gray des écritures binaires avant une étude théorique de cet ordre. La seconde partie concerne l'énumération des combinaisons de p éléments parmi n avant de faire le lien avec une représentation binaire et de trouver une application sur une restriction du problème du sac à dos. La mise en œuvre demandée nécessite la manipulation de liste et de vecteur. Le sujet est de longueur raisonnable, les meilleurs candidats ont traité l'ensemble des questions.

Analyse globale des résultats

Le sujet a été bien compris. Malheureusement, la formulation de certaines questions a pu légitimement troubler les candidats ; elles étaient involontairement ambiguës pour une programmation en `OCaml` ou `Caml-light` (que nous avons décidé de tolérer encore cette année pour ne pas pénaliser les doublants). De plus une indication était fausse. Nous sommes désolés de ces défauts, mais nous avons apprécié la capacité des candidats à s'adapter, soit en signalant et corrigeant l'erreur, soit en trouvant des stratégies de contournement raisonnables. Nous avons bien entendu accepté toutes les solutions cohérentes.

Les copies sont globalement lisibles et correctement présentées, même si certains persistent à écrire des codes sur plusieurs pages, et utilisent de nombreuses fonctions auxiliaires, sans expliquer leurs rôles et avec des noms de fonctions et de variables neutres, ce qui rend certains codes incompréhensibles en temps raisonnable.

Il faut bien sûr préciser le type des fonctions lorsqu'il n'est pas imposé, ou quand une fonction auxiliaire est écrite. La syntaxe `OCaml` est bien respectée, même si certains inventent des fonctions comme `List.make` ou continuent à mélanger des éléments `Python`, par exemple `for i in range`. Nous constatons toujours de grandes difficultés sur les références : oubli fréquent de `!`, syntaxes farfelues, création d'une référence dans une fonction récursive au lieu de la créer à l'extérieur puis de la modifier avec la fonction récursive. Lors des filtrages, certains oublient le cas de base ou retournent des messages d'erreur dans ce cas, ou encore renvoient des types différents selon les cas. Nous constatons également que de nombreux candidats ne savent pas identifier les situations de filtrage où le mot-clé `when` est indispensable. Ils confondent la variable, muette dans le filtrage, et sa valeur. Par exemple la fonction

```
let f x y = match x with
     y -> 1
    |_ -> 0 ;;
```

utilisée pour comparer des entiers renvoie toujours 1. Enfin, des confusions sur les rôles respectifs de `::` ou `@` et une utilisation souvent maladroite de `@`. De très nombreux candidats ne semblent pas mesurer le coût temporel de `q@[a]` pour l'ajout d'un élément `a` à la queue d'une liste `q`. Cependant, beaucoup de candidats maîtrisent bien tout cela et justifient leurs codes quand c'est nécessaire.

Commentaires sur les réponses apportées et conseils aux futurs candidats

Les premières questions ne sont pas claires puisqu'on demande de modifier la liste reçue en argument et de retourner un booléen. Retourner un couple `bool*list` a été une solution régulièrement adoptée, ce qui est très bien, mais dans ce cas, nous attendons que le candidat prenne en compte ce choix dans la question **Q2**. Par ailleurs, même si ce n'est pas usuel, le sujet demande d'*afficher* les n-uplets et non de créer une liste ou un tableau. Dans les questions **Q3** et **Q4**, on travaille sur des `'a list list`, la gestion du cas de base dans le filtrage et des concaténations n'est donc pas évidente et certains candidats ne prennent pas le temps de bien vérifier les types. Un nombre important de candidat, par un usage inapproprié de `@`, fusionnent les listes au lieu de garder des listes de listes. Nous avons également observé des incompréhensions sur ce qu'est une récursivité croisée et ce que cela induit en terme de complexité. À noter que certains candidats ont directement programmé à la question **Q4** une fonction « améliorée » au sens de la question **Q6**. Ils ont bien entendu été récompensés. Les questions suivantes, de la première partie, n'ont pas posé de difficulté particulière, mais certains candidats ne lisent pas bien la question et donc n'y répondent pas comme souhaité. On peut tout de même rappeler que, quand on utilise une démonstration par récurrence, il convient de bien vérifier l'initialisation, de poser l'hypothèse de récurrence et de démontrer l'hérédité, en utilisant l'hypothèse de récurrence.

La deuxième partie commence par un cas particulier. Il faut suivre l'avancement du sujet, il ne s'agit donc pas d'utiliser l'algorithme général de la question **Q17** pour traiter le cas de la question **Q1**4. Une erreur s'est glissée dans l'indication de la question **Q1**6 ; une grande partie des candidats a corrigé ce point. Nous avons récompensé ceux qui ont suivi l'indication. Par contre, attention, sauf quand on modifie le dernier élément, il faut modifier tous les éléments qui suivent l'indice à modifier. Il convient de bien faire attention aux types utilisés pour les questions **Q16** et **Q17** ; le sujet impose l'utilisation de tableaux. De plus, attention à l'utilisation de l'évaluation paresseuse :

```
while c.(!i + 1) = c.(!i) + 1 && !i < p - 1
```

n'équivaut pas à

```
while !i < p - 1 && c.(!i + 1) = c.(!i) + 1.
```

Comme toujours, il faut rappeler que l'épreuve est corrigée par des humains, qui peuvent faire preuve de tolérance sur des erreurs de syntaxe peu importantes, mais doivent pouvoir comprendre sans difficulté les codes, ce qui devient vite impossible quand on utilise des fonctions auxiliaires nombreuses, simplement appelées `aux_1` à `aux_n` et utilisant les mêmes noms de variables. Pourquoi ne pas appeler par exemple `nombre_1` une fonction auxiliaire qui calcule le nombre de « 1 » contenus dans une liste ? Et bien sûr il faut conserver les noms et notations du texte.

Conclusion

Le temps de formation en informatique est limité et les candidats doivent apprendre deux langages dont la philosophie est notablement différente. C'est un exercice difficile, et le jury mesure cette difficulté. Nous ne pouvons donc que conseiller la pratique sur machine, seul moyen d'acquérir de bons réflexes de programmation afin d'écrire des codes clairs et lisibles. Les démonstrations théoriques doivent être précises. Les réponses doivent être argumentées. Il faut lire avec soin le texte et analyser exactement ce qui est demandé.

Néanmoins, avec toutes ces difficultés, beaucoup de candidats ont un niveau satisfaisant et l'adaptabilité dont ils ont fait preuve pour corriger les imperfections du texte est une grande satisfaction pour le jury. Un nombre important de copies sont d'un excellent niveau, alors qu'il est très difficile d'écrire des codes sans compilateur. Les meilleurs ont rendu des copies pratiquement parfaites. Le jury félicite tous ces candidats, qui se sont ainsi impliqués dans l'option informatique.