Option Informatique

Présentation du sujet

Le sujet 2018 de l'option informatique s'intéresse à une résolution du jeu de société *Ricochet Robots*. La mise en œuvre demandée nécessite la manipulation de vecteurs, de listes et de structures construites. Une étude sur le parcours en largeur et quelques questions de complexité complètent le sujet. Le problème est découpé en quatre parties relativement indépendantes, mais il est nécessaire de bien lire les hypothèses pour comprendre les mouvements possibles, et d'analyser correctement comment les données sont représentées. Le choix d'un texte de longueur raisonnable permet aux meilleurs candidats d'aborder l'ensemble du problème.

Analyse globale des résultats

Le sujet a été globalement compris. Les meilleurs candidats ont pu traiter le problème en entier. À l'autre extrémité, il reste quelques copies très faibles, qui ne sont parfois même pas rédigées en Caml, de candidats qui n'ont sans doute pas travaillé du tout la matière. Les critiques générales sont malheureusement les mêmes d'année en année. Les signatures des fonctions Caml étaient imposées, les réponses doivent correspondre. La syntaxe Caml est parfois peu respectée. L'intérêt (ou non) et le bon usage des *références* n'est pas toujours compris. La notion de variable globale ou locale dans une fonction n'est pas maitrisée. Le filtrage est mal utilisé, certains candidats filtrent sur le nom de la variable cherchée au lieu de tester la valeur de l'expression filtrée. L'analyse de complexité n'est pas souvent justifiée, et n'est pas toujours conforme au code effectivement écrit. Par contre, l'utilisation des booléens était satisfaisante cette année. Beaucoup de candidats réécrivent plusieurs fois les mêmes fonctions, ce qui leur fait perdre du temps et complique la lecture en multipliant les fonctions auxiliaires, parfois sur plusieurs pages. Par exemple, le jury ne comprend pas l'intérêt des structures du type : `let f x y = let aux x y = ... in aux x y ;;`. Il est rare que la réponse à une seule question, surtout en tout début de problème, nécessite de nombreuses fonctions auxiliaires. Néanmoins, beaucoup de candidats ont des présentations agréables, avec changements de page corrects, une couleur différente pour les codes et les justifications ou démonstrations, des indentations, qui améliorent la lecture et des commentaires pertinents. Ces derniers ne sont pas nécessaires quand il s'agit d'expliquer un code très court, mais deviennent indispensables pour justifier la complexité d'un algorithme ou expliquer le bon fonctionnement de fonctions complexes en indiquant le rôle des fonctions auxiliaires par exemple.

Commentaires sur les réponses apportées et conseils aux futurs candidats

La première question du problème est l'écriture d'une *dichotomie*. Jon Bentley signalait dans *Programming Pearls* que « *A binary search program is notoriously hard to get right* ». Nous avons constaté que c'est toujours vrai. De plus, lors du calcul de complexité de la question **Q6.**, il fallait prendre en compte cette dichotomie. Pour le reste de la première partie, la difficulté principale pour les candidats réside dans la compréhension du sujet. Mouvement élémentaire d'un robot donné jusqu'à un obstacle et non d'une seule case, arrêt sur la case avant l'autre robot et non la même, copie de `mat_deplacements.(a).(b)` avant sa modification et non modification du tableau général, confusion entre les vecteurs et les listes, non respect des signatures imposées, etc.

La deuxième partie porte sur l'écriture de fonctions très classiques. Une majorité de candidats a traité ces questions de façon satisfaisante. Mais une importante fraction se perd en compliquant inutilement

le problème. Attention par exemple aux temps de calcul quand on a écrit le tri en passant par un accumulateur : le pire des cas est alors la liste croissante. Enfin le typage en Caml est strict : `insertion x q` et `insertion (x, q)` sont de signatures différentes et si `x` est de type `'a` et `q` une liste alors `[x]::q` est de type `'a list list`. Les fonctions de concaténation de liste (`@`) ou image miroir (`rev`) n'ont pas à être reprogrammées, mais elles doivent être utilisées avec précaution en raison de leurs temps de calcul non constants. Les candidats doivent bien comprendre la différence entre `x::q` et `q@[x]`.

La troisième partie porte sur les tables de hachage. Les principales erreurs proviennent d'une incompréhension de fond de l'intérêt de ces tables. Il y a alors des confusions entre la clé et l'élément. De plus, de nombreux candidats renoncent à utiliser les fonctions déjà écrites et aboutissent à des codes inutilement lourds. Enfin, les signatures ne sont pas respectées : des fonctions qui doivent être de type `unit`, renvoient des listes ou des tables.

La dernière partie propose une résolution du jeu grâce à un parcours en largeur d'un graphe. Les questions sur le sujet sont proches de celles posées l'année dernière, et malheureusement les difficultés observées sont les mêmes. Au lieu d'être démontrés, des éléments sont affirmés ou qualifiés d'« évidents » ou encore de résultats de cours, ce dont le jury ne doute pas, mais qui ne peut suffire à le convaincre que la notion est assimilée par le candidat. Les démonstrations doivent être complètes et précises à fortiori s'il s'agit d'un résultat du cours.

Pour finir, il faut encore et toujours redire que l'épreuve est corrigée par des humains. Le point positif est une certaine capacité de tolérance sur des erreurs de syntaxe peu importantes, mais le point négatif est une grande difficulté à comprendre un code appelant cinq ou six fonctions auxiliaires et définissant de multiples variables avec des noms non signifiants. Il importe donc d'écrire des codes clairs, mais également d'utiliser des notations courantes ou explicitées, et bien sûr de conserver celles du texte quand elles sont données.

Conclusion

Le temps de formation en informatique est limité dans cette classe et le jury mesure la difficulté imposée par l'apprentissage simultané de deux langages dont la philosophie est très différente. De ce fait, il faut absolument pratiquer sur machine pour acquérir les bons réflexes. Or l'horaire ne prévoit pas de travaux pratiques et les conséquences sont visibles pour certains qui n'ont sans doute programmé que sur papier. On ne peut que conseiller aux candidats de faire cet effort par eux-mêmes. La lisibilité des codes, la précision des démonstrations, la conformité des réponses avec les questions posées sont primordiales. Pour cela, il faut avoir des idées claires sur les bases du langage et une bonne capacité d'adaptation. Il faut également prendre le temps de bien lire et comprendre les implications des hypothèses et des indications qui sont données dans le texte.

Malgré les difficultés évoqués, beaucoup de candidats ont un niveau satisfaisant. Et un nombre important de copies sont d'un excellent niveau, alors qu'il est très difficile d'écrire des codes sans compilateur. Le jury félicite les candidats qui ont fait les efforts nécessaires pour maitriser ainsi la discipline.