ÉCOLE POLYTECHNIQUE FILIÈRE MP

CONCOURS D'ADMISSION 2000

COMPOSITION D'INFORMATIQUE

(Durée : 4 heures)

L'utilisation des calculatrices n'est pas autorisée

⋆ ⋆ ⋆

Avertissement. On attachera une grande importance à la clarté, à la précision et à la concision de la rédaction.

Introduction. Nous considérons un système commandé par un tableau de bord comportant N interrupteurs, chacun pouvant être baissé ou levé. On désire tester ce système (pour le valider ou pour effectuer une opération de maintenance) en essayant mécaniquement chacune des 2^N configurations possibles pour l'ensemble des interrupteurs. Le coût de cette opération, qu'elle soit réalisée par un opérateur humain ou par un robot, sera le nombre total de mouvements d'interrupteurs nécessaires. Nous supposons que chaque fois qu'un interrupteur est commuté le système effectue un diagnostic automatiquement et instantanément. Les interrupteurs sont indexés de 0 à $N-1$.

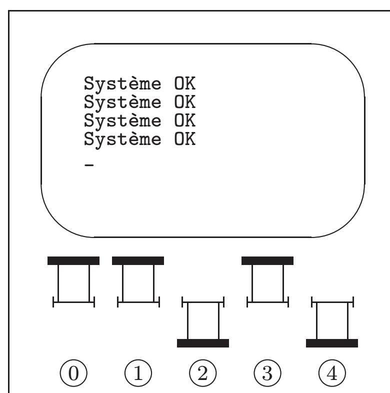

FIGURE 1. Pupitre de commande ; les interrupteurs 2 et 4 sont baissés.

Notations. Nous appelons *partie* un sous-ensemble fini de l'ensemble $\mathbf{N}$ des entiers naturels. Un élément d'une partie est appelé *indice*. La *différence symétrique* de deux parties P et Q est définie par :

$$P \vartriangle Q = (P \setminus Q) \cup (Q \setminus P) = (P \cup Q) \setminus (P \cap Q)$$

On vérifie facilement que la différence symétrique est commutative et associative, ce que l'on ne demande pas de démontrer. Pour tout entier n positif ou nul, nous notons $I_n = \{0, .., n-1\}$ l'ensemble des entiers inférieurs strictement à n.

Une partie sera représentée par une liste chaînée d'indices distincts apparaissant dans l'ordre croissant des entiers. On utilisera les types suivants :

Caml

```
type indice == int ;;
type partie == indice list ;;
```

Pascal

```
type indice = integer ;
type partie = ^noeud ;
noeud = record
  valeur : indice ; suivant : partie ;
end ;
```

Les candidats composant en Pascal, pourront utiliser le constructeur suivant :

```
function cree_partie (v : indice ; s : partie) : partie ;
var p : partie ;
begin new (p) ; p^.valeur := v ; p^.suivant := s ; cree_partie := p end ;
```

Première partie. Parties d'un ensemble

Question 1. Écrire la fonction `card` qui retourne le nombre d'éléments d'une partie.

Caml	Pascal
`value card : partie -> int`	`function card (p : partie) : integer ;`

Les candidats composant en Caml devront résoudre cette question sans faire appel à la fonction de bibliothèque `list_length`.

Question 2. Écrire la fonction `delta` qui réalise la différence symétrique de 2 parties. Le nombre d'opérations ne devra pas excéder $O(m+n)$, où m et n sont les cardinaux des arguments.

Caml	Pascal
`value delta : partie -> partie -> partie`	`function delta (p,q : partie) : partie ;`

Nous rappelons que dans toute liste chaînée de type `partie` les indices sont distincts et doivent apparaître dans l'ordre croissant des entiers.

Question 3. Application au problème des interrupteurs : à chacune des configurations possibles, nous associons la partie formée des indices des interrupteurs baissés.

Écrire un programme qui imprime la liste des indices d'interrupteurs à commuter pour passer d'une configuration à une autre.

Caml	Pascal
`value test : partie -> partie -> unit`	`procedure test (p,q : partie) ;`

Pour imprimer un entier `i` suivi d'un espace ou pour imprimer un saut de ligne, on pourra utiliser respectivement les instructions suivantes :

Caml	Pascal
`printf "%d " i ;`	`write (i,' ') ;`
`print_newline () ;`	`writeln ;`

Deuxième partie. Énumération des parties par incrément

À toute partie P, nous associons l'entier $e(P) = \sum_{i \in P} 2^i$ (avec la convention $e(\emptyset) = 0$). Nous définissons le *successeur* de P comme l'unique partie dont l'entier associé est $e(P)+1$.

Question 4. Écrire la fonction `succ` qui retourne le successeur d'une partie. Le nombre d'opérations ne devra pas excéder $O(l)$ où l est le plus petit indice absent dans la partie donnée en argument.

Caml	Pascal
`value succ : partie -> partie`	`fonction succ (p : partie) : partie ;`

Question 5. En application de ce mode d'énumération des parties, nous voulons réaliser le test de toutes les configurations d'interrupteurs. Au début et à la fin du test tous les interrupteurs seront levés.

a) Écrire un programme qui imprime la liste des indices des interrupteurs à commuter pour réaliser la totalité du test pour N interrupteurs et qui examine les configurations dans l'ordre défini par le successeur. L'argument de cette fonction sera l'entier N.

Caml	Pascal
`value test_incr : int -> unit`	`procedure test_incr (n : integer) ;`

b) Exprimer, en fonction de N, le nombre total d'interrupteurs à commuter pour réaliser le test de cette manière.

Troisième partie. Énumération des parties par un code de Gray

Nous notons $\langle u_0, \ldots, u_{l-1}\rangle$ une suite finie de l entiers. La concaténation de deux suites finies de longueur l et l' respectivement est une suite finie de longueur $l + l'$ définie par

$$\langle u_0, \ldots, u_{l-1}\rangle \odot \langle u'_0, \ldots, u'_{l'-1}\rangle = \langle u_0, \ldots, u_{l-1}, u'_0, \ldots, u'_{l'-1}\rangle.$$

La suite vide, notée $\langle\,\rangle$, est la suite de longueur 0. Une suite finie U est *préfixe* d'une autre suite finie V s'il existe une suite finie W telle que $V = U \odot W$ (autrement dit U est le début de V). Pour tout entier positif ou nul n, nous considérons la suite finie $T(n)$ de longueur $2^n - 1$ définie par

$$T(0) = \langle\,\rangle \text{ et } T(n+1) = T(n) \odot \langle n\rangle \odot T(n).$$

Nous avons par exemple $T(1) = \langle 0\rangle$, $T(2) = \langle 0, 1, 0\rangle$ et $T(3) = \langle 0, 1, 0, 2, 0, 1, 0\rangle$.

Pour tout entier i positif ou nul nous notons t_i le $(i+1)$-ème élément de $T(n)$ s'il existe. Puisque $T(n)$ est préfixe de $T(n+1)$, la suite $(t_i)_{i\geqslant 0}$ est définie sans ambiguïté. Enfin, nous posons $S_0 = \emptyset$ et pour tout entier i positif ou nul nous définissons l'ensemble $S_{i+1} = S_i \vartriangle \{t_i\}$.

Question 6.

a) Donner la valeur de $T(4)$.

b) Donner la valeur de S_i pour tout i inférieur ou égal à 15.

Question 7. Nous voulons montrer que les S_i peuvent être utilisés pour énumérer les parties de I_n, et ainsi résoudre notre problème d'interrupteurs.

a) Donner la valeur de S_{2^n-1} pour tout $n \geqslant 0$.

b) Montrer que pour tout $n > 0$ et tout $i < 2^n$, on a $S_{2^n+i} = S_i \vartriangle \{n-1, n\}$.

c) En déduire que pour tout $n \geqslant 0$ l'ensemble $\mathcal{P}_n = \{S_0, S_1, \ldots, S_{2^n-1}\}$ est l'ensemble des parties de I_n.

Question 8. Application au problème des interrupteurs. Comme dans la Deuxième partie, nous imposons que les interrupteurs soient levés au début et à la fin du test.

a) Écrire un programme s'inspirant des résultats de cette partie, qui imprime une liste d'indices d'interrupteurs à commuter pour réaliser la totalité du test. L'argument de cette fonction sera le nombre d'interrupteurs N.

Caml	Pascal
`value test_gray : int -> unit`	`procedure test_gray (n : integer) ;`

b) Quel est le coût du test avec cette méthode (c'est-à-dire le nombre total d'interrupteurs à commuter) ? Peut-on réaliser le test à un coût moindre ?

Question 9. Successeur de Gray. Pour tout $i > 0$, on note $\min(S_i)$ le plus petit élément de S_i.

a) Donner une expression de t_i en fonction de S_i pour i impair.

b) Écrire la fonction `gray` qui prend en argument une partie et retourne celle qui la suit immédiatement dans l'ordre défini par la suite $(S_i)_{i\geqslant 0}$.

Caml	Pascal
`value gray : partie -> partie`	`fonction gray (p : partie) : partie ;`

Quatrième partie. Système défaillant

Chaque interrupteur baissé active une composante du système, et un mauvais fonctionnement de l'alimentation électrique provoque une défaillance dès que plus de K interrupteurs sur les N sont baissés.

Question 10. Écrire un programme qui imprime une liste d'interrupteurs à commuter, de taille minimale, permettant de passer d'une configuration non défaillante à une autre sans provoquer de défaillance. Les arguments de ce programme seront la partie de départ et la partie cible.

Caml	Pascal
`value test_sur : partie -> partie -> unit`	`procedure test_sur (p,q : partie) ;`

Question 11. L'inverse d'une suite finie T est obtenue en prenant ses éléments dans l'ordre inverse, nous la notons $\widetilde{T}$. Soit $T(n,k)$ la suite définie pour tout $k \geqslant 1$ et pour tout $n \geqslant 1$ par

$$\begin{aligned} T(1,k) &= \langle 0 \rangle \\ T(n+1,1) &= T(n,1) \odot \langle n-1, n \rangle \\ T(n+1,k+1) &= T(n,k+1) \odot \langle n \rangle \odot \widetilde{T}(n,k) \end{aligned}$$

Soit k un entier strictement positif. Pour tout entier i positif ou nul nous notons $t_{k,i}$ le $(i+1)$-ème élément de $T(n,k)$ s'il existe. Puisque $T(n,k)$ est préfixe de $T(n+1,k)$, la suite $(t_{k,i})_{i\geqslant 0}$ est définie sans ambiguïté. Enfin, nous posons $S_{k,0} = \emptyset$ et pour tout entier i positif ou nul nous définissons l'ensemble $S_{k,i+1} = S_{k,i} \bigtriangleup \{t_{k,i}\}$.

a) Exprimer la longueur $l_{n,k}$ de $T(n,k)$ en fonction de n, de k et du nombre $s_{n,k}$ de parties de I_n de cardinal inférieur ou égal à k.

b) Montrer que pour tout $k \geqslant 1$ et tout $n \geqslant 1$ l'ensemble $\mathcal{P}_{n,k} = \{S_{k,0}, S_{k,1}, \ldots, S_{k,l_{n,k}}\}$ est l'ensemble des parties de I_n de cardinal inférieur ou égal à k.

Question 12. Écrire un programme qui affiche une liste de $l_{N,K}+1$ interrupteurs à commuter permettant de vérifier toutes les configurations non défaillantes sans provoquer de défaillance, en commençant et en finissant avec des interrupteurs tous levés. Les entiers N et K sont donnés en arguments.

Caml	Pascal
`value test_panne : int -> int -> unit`	`procedure test_panne (n,k : integer) ;`

Question 13. Montrer que le coût d'un test commençant et en finissant avec des interrupteurs tous levés et ne provoquant pas de défaillance ne peut pas être inférieur à $l_{N,K}+1$.

* *
*

Rapport de M. Luc MARANGET, correcteur.

De l'épreuve

Pour la première fois cette année, la composition d'informatique durait quatre heures. Le problème était un peu plus long que les années précédentes. En particulier, il contenait une quatrième partie réellement difficile, qui n'a été abordée que par une minorité de candidats.

La notation a donc été assise essentiellement sur les trois premières parties. Elle produit une moyenne de 11,6, un écart type de 3,4 et une répartition classique des notes.

$0 \leq N < 4$	1%
$4 \leq N < 8$	11%
$8 \leq N < 12$	42%
$12 \leq N < 16$	36%
$16 \leq N \leq 20$	10%

En tout, 215 copies ont été corrigées, dont 190 rédigées en Caml et 25 en Pascal. Une fois encore, le faible nombre de candidats qui composent en Pascal ne permet pas tirer de réelles conclusions.

Toutefois, on peut remarquer que les toutes meilleures copies (un candidat a eu 20, trois ont eu 19) sont rédigées en Caml, qu'il y a une légère sur-représentation des mauvaises copies chez les adeptes de Pascal, mais que cet effet est compensé en partie par une meilleure réussite des copies moyennes chez ces mêmes adeptes. Bien malin qui pourra en tirer des enseignements sur l'impact du choix du langage sur les chances au concours des candidats.

Toutefois, à titre personnel, je ne peux que conseiller aux candidats et professeurs d'adopter Caml, bien plus représentatif de la programmation moderne que Pascal.

Du problème

Le code de Gray permet d'énumérer les parties d'un ensemble donné en ajoutant ou supprimant un seul élément à chaque fois. Il existe quelques applications pratiques, justement dans le cadre d'algorithmes d'énumération, de compteurs rapides ou à consommation constante. Mais on peut dire que l'esprit de ce sujet était plutôt celui d'un exercice.

Cependant le problème fait une présentation récursive du code de Gray, que je crois assez inhabituelle. Par conséquent, ce problème testait fortement les aptitudes des candidats face à la récursion. Et on peut dire que la récursion est comprise, souvent maîtrisée. Toutefois elle effraie encore un peu et dès qu'une programmation itérative devient possible (questions **3** et **4** par exemple) les candidats y ont recours.

Pourtant, à mon avis, une solution récursive est souvent plus simple à programmer sans

erreur, surtout sur papier. En effet, les programmes récursifs se déduisent souvent plus directement d'une propriété à garantir que les programmes itératifs.

Bien des candidats connaissaient manifestement le code de Gray. Cela n'a été un réel avantage que pour ceux qui ont su prendre le point de vue de l'énoncé. En particulier, j'ai vu de nombreuses propriétés non démontrées, voire des formules complexes reposant sur la décomposition en base deux. Or, il est difficile d'arriver au bout des preuves et des programmes à l'aide de ce point de vue. Il faut donc en conclusion se méfier face à un problème que l'on connaît déjà et lire attentivement l'énoncé, des « connaissances » sur le sujet traité n'apportant aucun point en tant que telles.

Enfin, la rédaction de l'énoncé était particulièrement dense. C'est je crois un avantage pour les candidats qui peuvent identifier plus facilement ce qu'il leur est demandé. Cette densité devrait faciliter la lecture de l'énoncé. Pourtant, la phrase « Au début et *à la fin* du test tous les interrupteurs seront levés. » est parfois passée inaperçue, tout comme cette autre : « Pour tout entier i positif *ou nul* nous notons t_i, le *(i+1)-ème* élément de $T(n)$. ». J'insisterais donc, cette année encore, sur l'importance de la lecture des définitions de l'énoncé.

Programmation

On note trois catégories de difficultés.

Tout d'abord, les programmes des copies pèchent souvent par manque de simplicité. Souvent, une réflexion préalable aurait permis aux candidats d'écrire des programmes plus courts et plus simples, et par là même, plus susceptibles d'être justes. Par exemple, il y a un rapport entre structures de données et programmes, et il n'est pas très astucieux d'afficher une liste à l'aide d'une boucle `for` et d'une fonction d'accès au i-ème élément d'une liste.

Ensuite, un souci excessif de l'efficacité conduit parfois les candidats à écrire des programmes trop longs ou trop compliqués.

Enfin, il convient de se souvenir qu'il s'agit aussi d'une épreuve de programmation.

Ainsi, une puissance de deux se calcule, il ne suffit par d'écrire 2^n dans sa copie.

En fait, les copies révèlent souvent le manque d'expérience de la programmation des candidats. S'il n'est pas humainement possible d'acquérir cette expérience en une ou deux années, les futurs candidats peuvent y tendre en prenant le parti, lors de leur formation, de rechercher les solutions les plus simples et les plus élégantes.

Ils auront aussi intérêt à examiner de près le rapport entre les mathématiques et la programmation : la spécification de propriétés à garantir aide à programmer juste, mais les programmes sont effectifs et réalisent en dernière analyse un calcul qui est plus qu'une affirmation d'existence.

Rapport détaillé

Dans le rapport qui suit est indiqué, pour chaque question, le pourcentage des candidats qui ont obtenu au moins la moitié des points.

Question 1. [96 %] Les quelques erreurs rencontrées sont le fait de candidats distraits, ne sachant pas programmer, ou effrayés par la liste vide. Rappelons donc, qu'en Pascal par exemple, l'opération interdite sur `nil` est le déréférencement, ainsi le test `p=nil` est légitime.

Question 2. [89 %] C'est presque une question de cours, une variante de la classique fusion de deux listes triées. Je suis satisfait de constater que presque tous les candidats, *y compris les candidats composant en Pascal* proposent des solutions récursives, sur ce point le cours est assimilé. Je ne suis pas impressionné par la performance réalisée par les candidats qui adoptent un point de vue itératif, car je trouve ça bien trop compliqué.

Je suis plus étonné de voir qu'un nombre significatif des copies proposent des solution avec récursion terminale, c'est tout à fait inutile ici.

C'est même un peu dangereux, les risques de confusion au moment de renverser la liste résultat étant bien réels.

Enfin, les solutions récursives terminales qui construisent directement une liste résultat triée par `l @ [x]` pour ajouter l'élément `x` à la fin de la liste `l` sont en fait quadratiques et ne répondent qu'imparfaitement à la question.

Or, les bons programmes sont dans l'ordre justes, simples et raisonnablement efficaces.

Les problèmes liées par exemple à la taille de la pile des appels ne se posent normalement pas dans le cadre d'une composition sur papier. Bien plus, il en est souvent de même en pratique.

Enfin, l'énoncé ne demande pas de donner une preuve de la fonction ou de sa complexité. La contrainte de complexité ne servant qu'à rendre obligatoire une réponse du style souhaité par les concepteurs du problème.

On peut ne rien dire (un code clair et classique parlant de lui-même) ou se contenter de donner un ou deux argument importants. Les candidats qui donnent de longues explications perdent du temps.

Question 3. [86 %] Il suffit d'afficher la liste rendue par la fonction de la question précédente. La question testait donc tout simplement la capacité des candidats à écrire une fonction qui affiche une liste d'entiers. Les erreurs relevées sont souvent des inattentions, dont la plus grave est l'oubli d'un appel récursif sur la queue de la liste à imprimer ou l'oubli d'une affectation `p := p^.suivant` dans une boucle. Il est dommage de perdre des points sur une question aussi simple.

Question 4. [67 %] J'aime bien cette question dont la solution est courte et pas tout à fait évidente. On se rend rapidement compte qu'il s'agit de coder une addition (plus

exactement un incrément) avec retenue en base deux. Toutefois, le codage proposé (liste des indices des bits à 1, et non pas liste des chiffres binaires) n'est pas très usuel et il convient de faire attention.

La plupart des copies abordent correctement cette question, les erreurs les plus graves proviennent d'une incompréhension du codage proposé, ou d'un oubli systématique du cas de la liste vide. Trop de candidats omettent de vérifier qu'une liste `p` n'est pas vide (`p <> nil` ou `p <> []`), avant d'en extraire le premier élément (`p^.valeur` ou `hd p`).

Une fois encore, on note de longs développement en plus du code, les bonnes copies les remplacent avantageusement par des assertions données en commentaire du code proposé. Ici, on peut concevoir une fonction intermédiaire qui à l'entier n et à la partie P, associe la partie Q telle que $e(Q) = e(P) + 2^n$, et que l'on appelle avec l'argument initial $n = 0$.

Question 5. [57 %] C'est l'analogie entre représentations binaires des entiers et parties de I_N qui suggère ce mode d'énumération des parties. Il s'agit donc d'utiliser la fonction de la question précédente pour énumérer les parties de $\emptyset$ ($e(\emptyset) = 0$) à I_N ($e(I_N) = 2^N - 1$), puis de revenir à $\emptyset$. L'oubli de cette dernière étape a coûté bien des points à bien des candidats. En effet, le programme qui répond complètement à la question posée étant significativement plus difficile à concevoir que celui qui n'y répond que partiellement, l'oubli n'est donc pas bénin.

a) Il convient de savoir quand arrêter l'énumération des parties, le plus simple est de s'arrêter lorsque le *successeur* de P est $\{N\}$.

Un codage par fonction récursive résout bien des difficultés. Une boucle `while` qui teste que la partie atteinte n'est pas encore de cardinal n (`card(p) < n`), ou que son successeur ne débute pas par n sont des solutions absolument équivalentes.

Le codage de l'énumération par une boucle `for` dont l'indice est plus ou moins l'entier représenté par la partie atteinte conduit souvent à des erreurs de bornes.

On doit alors fixer clairement le lien entre l'indice de boucle i et la partie atteinte P, partie qui change entre l'entrée et la sortie d'une itération de boucle.

Classiquement (et c'est valable pour toutes les méthodes) il est prudent de vérifier que le programme proposé fait bien ce que l'on attend de lui en l'exécutant dans sa tête pour $N = 1$ par exemple. Notons au passage que presque aucun candidat n'a pris ses précautions pour le cas $N = 0$ ce qui est en définitive assez normal compte tenu de la nature du problème. Il n'en a donc pas été tenu compte.

b) Il y a une méthode très astucieuse, qui consiste dans un premier temps à compter les mouvements d'interrupteurs du haut vers le bas (un à chaque étape d'incrément), puis à remarquer qu'il y a autant de mouvements en sens inverse.

Aucun candidat n'a vu cette astuce. Ceux qui procèdent intuitivement ont plutôt dénombré les mouvements interrupteur par interrupteur, car il est assez clair que l'interrupteur d'indice 0 change à toutes les étapes, celui d'indice 1 une fois sur deux, etc... jusqu'à

l'interrupteur d'indice $N-1$ qui change deux fois. Cette intuition devait se confirmer par un raisonnement précis.

Sinon (et sans doute plus sûr), on pouvait trouver des relations de récurrence, qu'il convient de bien expliquer, ou sommer directement les interrupteurs bougés à chaque étape. De façon générale les candidats investissent parfois leur énergie plus dans les calculs que dans la justification de leurs dénombrements. Une attitude qui se révèle pénalisante quand les calculs sont faux et ne sont pas expliqués. J'ai dû alors remonter au dénombrement initial pour me rendre compte qu'il était donné tel quel, parfois faux, parfois juste, souvent insuffisamment expliqué. En particulier, la question de savoir si l'étape de remise à zéro de tous les interrupteurs était consciemment dénombrée ou laissée de côté pour la fin a eu une grosse influence sur la notation.

Il faut aussi noter que les solutions de récurrences faciles (par ex. $C(N+1) = 2C(N)+2$, à transformer en $C(N+1)+2 = 2(C(N)+2)$) sont parfois un peu hésitantes, avec parfois un passage par les sommations.

Question 6. [99 %] Il suffit d'un peu de méthode, on peut quand même en profiter pour se familiariser avec une importante convention de l'énoncé, à savoir que les indices des t_i commencent à zéro.

Question 7. [63 %] Le but de cette question est de prouver que l'ensemble des S_i, pour i variant de 0 à $2^n - 1$ est l'ensemble des parties de $I_n = \{0, 1, \dots n-1\}$.

Chaque sous-question est une étape facile de la preuve. Suivre l'esprit du problème revient à donner une preuve la plus simple possible à chaque fois en évitant de prouver (voire de présenter comme évidentes) des propriétés plus complexes que celles demandées. Or, la seule propriété vraie par construction est celle-ci :

$$\begin{cases} t_{2^n-1} = n & \text{pour } n > 0 \\ t_{2^n+k} = t_k & \text{pour } 0 \leq k < 2^n - 1 \end{cases} \tag{1}$$

(Propriété qui d'ailleurs évoque plus une translation qu'une symétrie.)

Les sous-questions **a** et **b** ont donné lieu à des développement parfois bien longs et à des notations du style $S_{2^n-1} = \Delta_{k=0}^{k=2^n-2}\{t_k\}$ que l'on ne pouvait introduire qu'après avoir constaté l'associativité de la différence symétrique, voire sa commutativité. Aucune des deux questions ne nécessitait de récurrence sur n. Pour la sous-question **c**, la récurrence était la solution la plus simple et les cas de base étaient à vérifier pour $n = 0$ et $n = 1$. En effet, l'étape d'induction de n à $n+1$ se résolvait en utilisant **b**, qui ne s'appliquait que pour $n \geq 1$.

Dans cette même sous-question, de nombreux élèves déduisent l'égalité $\text{card}(\mathcal{P}_n) = 2^n$ de l'écriture $\mathcal{P}_n = \{S_0, S_1, \dots, S_{2^n-1}\}$, alors que l'on pouvait seulement déduire l'inégalité $\text{card}(\mathcal{P}_n) \leq 2^n$.

Dans l'ensemble de cette question j'ai rencontré de très nombreuses confusions d'indices et de bornes. Ce type d'erreur n'a pas été exagérément pénalisé, surtout lorsque la clarté

du raisonnement ou la présence de schémas révélaient que le candidat avait les idées claires. Il n'en a bien entendu pas été de même dans les programmes qui vont suivre. Plus généralement, les candidats ont du mal (et c'est bien normal) à rédiger les preuves. Peu adoptent la stratégie de rédiger des preuves courtes mettant en valeur les idées et les étapes importantes.

Question 8. [47 %] Selon la question précédente, il s'agit simplement d'afficher la suite $T(n)$, suivie de $n-1$.

a) Bien des candidats entraînés par l'exemple de l'énumération par incrément construisent les S_i, et affichent t_i par `test` S_i S_{i+1}. Dans le meilleur des cas cette construction est locale à une boucle de parcours des t_i. Dans le pire des cas, une liste ou un tableau des S_i sont construits parfois au prix d'un calcul complet d'un préfixe de la suite T à chaque étape. Ces dernières solutions, souvent fausses parce que compliquées, ont aussi été pénalisées pour elles-mêmes. En effet, elles ont un coût tout à fait prohibitif, mais surtout elles révèlent une mauvaise analyse de ce qui doit être programmé. Rappelons que l'avertissement initial « la concision est appréciée » est particulièrement pertinent dans le cas des programmes eux-mêmes.

Bien plus, pour afficher les t_i, il n'est pas besoin de construire explicitement une structure de données les contenant. On peut programmer en s'inspirant directement de la définition récursive de l'énoncé, rendue effective comme $T(n) = T(n-1) \odot \langle n-1 \rangle \odot T(n-1)$, puis en remplaçant « $\langle n-1 \rangle$ » par l'affichage de cet entier. Cette solution est rarissime, la plupart des copies, même les meilleures, construisent explicitement une liste (ou un tableau) des t_i. À ce stade les erreurs d'indices sont extrêmement nombreuses, cela va du plus grave (calculer `T(n)` par `T(n) @ [n] @ T(n)` en notation Caml), à la faute d'indice moins lourde de conséquence (calculer `T(n)` par `T(n-1) @ [n] @ T(n-1)`).

b) La réponse étant « évidente », la question testait la capacité des candidats à donner l'argument qui prouve une évidence, le choix des mots devenait alors crucial. Les meilleures justifications font référence au code écrit à la question précédente pour la première partie de la question et à un raisonnement simple pour la seconde, il était alors opportun de rappeler que les S_i sont deux à deux distincts.

Question 9. [19 %] On a $t_i = \min(S_i) + 1$ pour i impair, et $t_i = 0$ pour i pair. On peut le découvrir sur les exemples du **6b** ou déjà le savoir. Cette formule étant en effet une présentation courante du code de Gray.

a) Il faut ensuite le prouver. On peut procéder élégamment en montrant que la suite définie par :

$$\begin{cases} m_{2p} = 0 \\ m_{2p+1} = min(S_{2p+1}) + 1 \end{cases}$$

obéit aux conditions de définition de la suite de Gray (1).

Finalement assez peu de candidats abordent la preuve de cette formule, pourtant sans surprises. Les preuves proposées sont souvent convenables quoiqu'un peu rapides dans la justification de $\min(S_i \,\Delta \{n-1, n\}) = min(S_i)$. Les preuves par décomposition en

puissances de deux me semblent difficiles surtout quand on considère qu'elle reviennent à identifier deux suites et que la définition récursive de la suite de Gray est de loin la plus commode.

b) Pour répondre à l'aide de la formule du **a** il faut, à partir d'une partie S retrouver la parité de son indice dans l'énumération de Gray. Or, il se trouve que les parités du cardinal de S_i et de i sont identiques, il convenait d'expliquer pourquoi.

Comme je m'y attendais un peu, j'ai constaté une variété considérable de tests de parité faux (le plus simple est d'utiliser l'opérateur reste de la division euclidienne, `mod` en Caml et en Pascal). Les plus surprenants traduisent un tour d'esprit assez fréquent chez les candidats qui composent en Caml et surestiment le pouvoir de leur langage, il s'agit de `if x = 0 mod 2 then ...` et de `match x with 2*p ->...`.

Question 10. [45 %] Une question plutôt facile en fait mais sa place dans le cours du problème a donné lieu à une correction rigoureuse. En particulier les solutions devaient être conceptuellement justes, d'où la nécessité d'un code clair et structuré et de quelques explications. Cette démarche facilite certes la correction, mais permet aussi au candidat d'éviter les erreurs stupides. On pouvait résoudre cette question en programmant la différence ensembliste, puisqu'il suffit, pour commuter les interrupteurs qui doivent l'être, de lever d'abord les interrupteurs qui doivent l'être (i.e., ceux qui sont baissés au départ et levés à l'arrivée), puis de baisser ceux qui doivent l'être. Quelques candidats oublient d'afficher les listes comme demandé dans l'énoncé.

Question 11. [2 %] On attaque les question authentiquement difficiles du problème. Ici des solutions partielles dégageant de bonnes intuitions pouvaient devenir rénumératrices.

a) Une attitude féconde devant cette question est de commencer par donner les définitions par récurrence des suites l et s, puis de les identifier par des manipulations diverses. On retrouvait facilement la relation de récurrence des C_n^p, en compagnie d'autres relations pour les valeurs initiales des paramètres n et k. Il convenait de ne pas oublier ces cas, sinon on prouvait n'importe quoi (que l'on songe à la suite définie par $a_{1,0} = a_{1,1} = 0$ et $a_{n+1,p+1} = a_{n,p} + a_{n,p+1}$ par exemple).

Il est remarquable qu'une petite dizaine de candidats aient changé les définitions de l'énoncé pour arriver au résultat $l_{n,k} = s_{n,k} - 1$, satisfaisant pour eux par analogie avec les questions précédentes. Ces candidats ont supposé que la notation $\{S_{k,0}, S_{k,1}, \ldots, S_{k,l_{n,k}}\}$ de la question **b** entraîne que les $S_{k,i}$ sont deux à deux distincts. Or, il n'en est rien, comme on s'en rend compte facilement dans le cas $k = 1$. En effet pour répondre au problème d'énumérer les parties à au plus un interrupteur baissé, on devra certainement passer plusieurs fois par l'état « tous levés ». Cette simple vérification aurait dû permettre à ces candidats de porter le bon jugement sur l'énonce : notation peut-être un peu compacte, mais certainement pas faute dans une définition importante.

b) Les deux points-clefs de la démonstration sont d'une part l'introduction de $S_{k,l_{n,k}} = \{n-1\}$ dans l'hypothèse d'induction et l'utilisation de l'inversion de la suite $T(n, k)$. Les rares candidats qui les évoquent engrangent des points.

Question 12. [7 %] Cette question est très semblable à la question **8a**, la notation a porté sur l'expression correcte de la suite T (attention aux indices !) ainsi que sur la présence justifiée d'un mouvement d'interrupteur pour retourner à la position « tous levés ». Cette dernière étape peut se faire même si l'on n'a pas remarqué que $S_{k,l_{n,k}}$ vaut $\{n-1\}$, en calculant les $S_{k,i}$ au fur et à mesure et par « `test S []` », à la fin de l'itération.

Question 13. [0 %] Personne n'a répondu à cette question, qui demande une astuce difficile à produire en fin d'épreuve. Il faut d'abord penser à regrouper les parties par cardinal, en remarquant que les transitions « montantes » et « descendantes » sont en nombre égal. Ainsi, le nombre total de parties atteintes est deux fois la somme du nombre des transitions montantes. D'autre part, pour un cardinal donné p, la somme des nombre de transitions montantes issues des parties de cardinal p et des parties de cardinal $p-1$ est minorée par C_n^p. Il reste ensuite à sommer $C_n^k + C_n^{k-2} + \cdots$.