Composition d'informatique n°6

Sujet 1 : algorithmique et programmation (Durée : 4 heures)

L'utilisation de la calculatrice **est autorisée** pour cette épreuve.

⋆ ⋆ ⋆

Le sujet contient deux parties indépendantes. La première partie s'intéresse à un algorithme de clustering en dimension 1. La deuxième partie présente le théorème de Curry-Howard qui fait le lien entre la déduction naturelle et le typage en OCaml.

1 Clustering en dimension 1

On souhaite mettre en place du soutien différencié au travail dans une classe de lycée. Pour ce faire, on souhaite séparer les élèves en plusieurs groupes de niveaux homogènes pour leur proposer des exercices adaptés à leur niveau. On dispose pour cela des notes des élèves et on veut les regrouper selon la similitude de leurs notes.

Formellement, on dispose d'un ensemble E de N réels $x_0 \leqslant x_1 \leqslant \ldots \leqslant x_{N-1}$. On cherche à déterminer une partition de $[\![0, N-1]\!]$ en $K \leqslant N$ sous-ensembles $\mathcal{P} = \{C_0, C_1, \ldots, C_{K-1}\}$ non vides tels que le score

$$S(\mathcal{P}) = \sum_{i=0}^{K-1} \sum_{j \in C_i} (x_j - \mu_i)^2$$

soit minimal, où $\mu_i = \dfrac{1}{|C_i|} \displaystyle\sum_{j \in C_i} x_j$ est la moyenne des éléments correspondant à la classe C_i. Autrement dit, on veut minimiser la somme des carrés des écarts de chaque élément à la moyenne de sa classe.

Par exemple, pour $E = \{1, 2, 3, 5, 8, 10, 14, 15, 18\}$, une solution optimale pour $K = 3$ est donnée par la partition suivante :

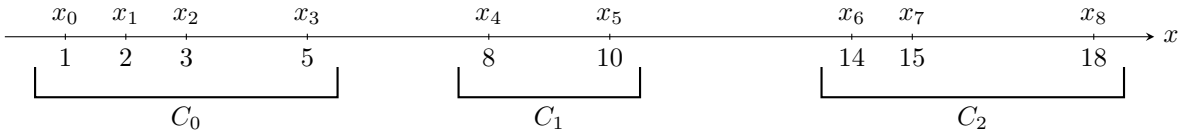

FIGURE 1 – Une partition optimale $\mathcal{P} = \{\{0, 1, 2, 3\}, \{4, 5\}, \{6, 7, 8\}\}$ de $[\![0, 8]\!]$ pour $K = 3$ et l'ensemble $E = \{1, 2, 3, 5, 8, 10, 14, 15, 18\}$, de score environ $19,42$.

Si $\{C_0, C_1, \ldots, C_{K-1}\}$ est une partition solution du problème, on remarque qu'on peut supposer sans perte de généralité que les classes sont rangées par ordre croissant, c'est-à-dire que pour $i < i'$ et $j \in C_i$, $j' \in C_{i'}$, alors $x_j \leqslant x_{j'}$. On supposera cette hypothèse vérifiée pour l'ensemble des partitions du problème.

1.1 Préliminaires

Question 1 Comment trouver une solution au problème lorsque $K = N$? Lorsque $K = N - 1$? Justifier.

Question 2 Appliquer l'algorithme des K-moyennes sur l'ensemble E de l'exemple précédent. On prendra $K = 3$, et on initialisera les barycentres à $b_0 = 1$, $b_1 = 8$ et $b_2 = 10$. On donnera les détails des étapes de calculs jusqu'à convergence de l'algorithme.

On accepte une représentation graphique pour la description des étapes de calcul.

On cherche à écrire une fonction `void tri(double* tab, int n)` qui trie un tableau `tab` de taille `n` en C. Pour ce faire, on propose les fonctions suivantes :

```c
void fusion(double* tab1, int n1, double* tab2, int n2, double* tab){
    int i1 = 0;
    int i2 = 0;
    for (int i=0; i<n1 + n2; i++){
        if (tab1[i1] <= tab2[i2]){
            tab[i] = tab1[i1];
            i1++;
        } else {
            tab[i] = tab2[i2];
        }
    }
}

void tri(double* tab, int n){
    if (n < 1) return;
    int n1 = n / 2;
    int n2 = n - n / 2;
    double* tab1 = malloc(n1 * sizeof(double));
    double* tab2 = malloc(n2 * sizeof(double));
    for (int i=0; i<n; i++){
        if (i < n1) tab1[i] = tab[i];
        else tab2[i - n1] = tab[i];
    }
    tri(tab1, n1);
    tri(tab2, n2);
    fusion(tab1, n1, tab2, n2, tab);
}
```

Question 3 Le code précédent comporte plusieurs (moins de 5) erreurs. Préciser les lignes où apparaissent ces erreurs et un moyen de les corriger. On demande de **NE PAS** recopier tout le code.

Pour la suite de cette partie, on supposera que le tableau `E` représentant un ensemble E sera toujours trié.

Pour $E = \{x_0, \ldots, x_{N-1}\}$ et $0 \leqslant i < j \leqslant N$, on note $S_{\text{emc}}(i, j)$ (pour « somme des écarts à la moyenne au carré ») la valeur

$$S_{\text{emc}}(i, j) = \sum_{k=i}^{j-1} (x_k - \mu)^2$$

où μ est la moyenne des éléments de $\{x_i, x_{i+1}, \ldots, x_{j-1}\}$.

Question 4 Écrire une fonction `double moyenne(double* E, int i, int j)` qui prend en argument un tableau E de N valeurs $\{x_0, \ldots, x_{N-1}\}$ triées et deux indices $0 \leqslant i < j \leqslant N$ et renvoie la moyenne des éléments de $\{x_i, x_{i+1}, \ldots, x_{j-1}\}$.

Question 5 Écrire une fonction `double somme_emc(double* E, int i, int j)` qui prend en argument une liste E de N valeurs triées et deux indices $0 \leqslant i < j \leqslant N$ et calcule et renvoie $S_{\text{emc}}(i, j)$.

On représente une partition $\mathcal{P} = \{C_0, C_1, \ldots, C_{K-1}\}$ de $[\![0, N-1]\!]$ par un tableau `P` de taille K telle que `P[i]` est le plus petit élément de C_i. Avec l'hypothèse faite précédemment sur les C_i, le tableau `P` doit nécessairement être trié. On remarque, de plus, que `P[0]` vaut toujours 0. Par exemple, la partition $\{\{0, 1, 2, 3\}, \{4, 5\}, \{6, 7, 8\}\}$ donnée dans l'exemple de la figure 1 est représentée par `{0, 4, 6}`.

Question 6 Dans cette question, on suppose $N = 9$. Quelle est le tableau `P` associé à la partition $\mathcal{P} = \{\{0, 1, 2\}, \{3, 4\}, \{5, 6, 7\}, \{8\}\}$? Quelle est la partition associée au tableau `{0, 1, 4, 5}` ?

Question 7 Écrire une fonction `double score(double* E, int N, int* P, int K)` qui prend en argument un ensemble `E` de taille N et une partition `P` de $[\![0, N-1]\!]$ de taille K et renvoie le score $S(\mathcal{P})$ tel qu'il a été défini précédemment.

Question 8 En déduire une fonction `int* clustering3(double* E, int N)` qui prend en argument un ensemble `E` de taille $N \geqslant 3$ et renvoie une partition `P` de $[\![0, N-1]\!]$ de taille $K = 3$ de score minimal.

Question 9 Quelle est la complexité temporelle de la fonction précédente ? Justifier.

1.2 Clustering hiérarchique ascendant

On cherche dans cette sous-partie à calculer une solution pas nécessairement optimale par une approche de clustering hiérarchique ascendant (CHA).

Question 10 Écrire une fonction `int classes_plus_proches(double* E, int N, int* P, int K)` qui prend en argument un ensemble `E` de taille N et une partition `P` de $[\![0, N-1]\!]$ de taille $K > 1$ et renvoie un indice i_{opt} tel que $C_{i_{\text{opt}}}$ et $C_{i_{\text{opt}}+1}$ sont les classes les plus proches, c'est-à-dire celles qui ont leurs moyennes les plus proches. En cas d'égalité, on choisira l'indice i_{opt} minimal.

Question 11 Écrire une fonction `int* fusion_classes(int* P, int K, int iopt)` qui prend en argument une partition `P` de $[\![0, N-1]\!]$ de taille K et un indice $0 \leqslant i_{\text{opt}} < K-1$ et renvoie une partition de $[\![0, N-1]\!]$ de taille $K-1$ où les classes d'indices i_{opt} et $i_{\text{opt}}+1$ ont été fusionnées.

Question 12 En déduire une fonction `int* CHA(double* E, int N, int K)` qui calcule et renvoie une partition de taille K d'un ensemble E selon l'algorithme de clustering hiérarchique ascendant.

Question 13 Déterminer la complexité temporelle de la fonction `CHA` en fonction de N et de K.

Question 14 Montrer par un exemple bien choisi que l'algorithme de clustering hiérarchique ascendant ne permet pas de résoudre le problème de manière optimale.

On pourra prendre $N = 4$ et $K = 2$.

1.3 Solution optimale en programmation dynamique

Pour $n \in [\![0, N]\!]$ et $k \in [\![1, K]\!]$, on note $D(n, k)$ le score minimal possible d'une partition de $\{x_0, \ldots, x_{n-1}\}$ en k classes non vides.

Question 15 Que vaut $D(n, k)$ lorsque $k = 1$?

Question 16 Montrer que pour $n > 0$ et $k > 1$, $D(n, k) = \min_{i=k-1}^{n-1} (D(i, k-1) + S_{\text{emc}}(i, n))$.

Question 17 En déduire une fonction `double clustering_dynamique(double* E, int N, int K)` qui calcule le score minimal possible d'une partition de E en K classes non vides.

Question 18 Déterminer la complexité temporelle de la fonction précédente.

Question 19 Expliquer en français comment modifier la fonction précédente pour qu'elle renvoie une partition optimale plutôt que le score minimal. On ne demande pas de coder cette solution.

On cherche à améliorer la complexité temporelle de la solution précédente en effectuant des précalculs. Pour $0 \leqslant i < n \leqslant N$, notons $\mu(i,n)$ la moyenne des éléments de $\{x_i, \ldots, x_{n-1}\}$. On admet la formule suivante :

$$S_{\text{emc}}(0, n+1) = S_{\text{emc}}(0,n) + \frac{n}{n+1}(x_n - \mu(0,n))^2$$

Question 20 Décrire en français ou pseudo-code un moyen de calculer l'ensemble des $S_{\text{emc}}(i,n)$ pour $0 \leqslant i < n \leqslant N$ en complexité $\mathcal{O}(N^2)$. On détaillera les formules de récurrence utilisées.

2 Correspondance de Curry-Howard

On considère un ensemble fini de variables $\mathcal{V} = \{x_0, x_1, \ldots, x_{n-1}\}$, avec $n \in \mathbb{N}$. On définit l'ensemble des formules propositionnelles $\mathcal{F}$ de variables $\mathcal{V}$ par induction par :

- $\bot \in \mathcal{F}$;
- pour $x \in \mathcal{V}$, $x \in \mathcal{F}$;
- si $A, B \in \mathcal{F}$, alors $A \wedge B$, $A \vee B$ et $A \to B$ sont dans $\mathcal{F}$.

On remarque en particulier que la négation ne fait pas partie de la définition par induction des formules.

On rappelle en annexe les règles d'inférence des logiques minimale, intuitionniste et classique. On notera $\Gamma \vdash_m A$, $\Gamma \vdash_i A$ et $\Gamma \vdash_c A$ pour indiquer qu'un séquent $\Gamma \vdash A$ est prouvable en logique minimale, intuitionniste et classique respectivement.

On admet et on pourra utiliser le fait que la logique classique est correcte et complète pour la sémantique booléenne usuelle.

2.1 Premières preuves

Pour $A \in \mathcal{F}$, on définit $\neg A$ par $\neg A = A \to \bot$. Il s'agit de sucre syntaxique, donc ces formules sont bien identiques et pas seulement sémantiquement équivalentes.

Question 21 Montrer que les règles suivantes sont admissibles en logique minimale.

$$\frac{\Gamma, A \vdash \bot}{\Gamma \vdash \neg A}\ \neg_i \quad \text{et} \quad \frac{\Gamma \vdash \neg A \qquad \Gamma \vdash A}{\Gamma \vdash \bot}\ \neg_e$$

On pourra les utiliser librement par la suite.

Dans une formule $A \to B \to C$, la priorité des évaluations est donnée de droite à gauche. Ainsi, cette formule s'interprète comme $A \to (B \to C)$.

Question 22 Montrer que $\vdash_m A \to (A \to B) \to B$, c'est-à-dire que $\vdash A \to (A \to B) \to B$ est prouvable en logique minimale.

Question 23 Montrer que $\dfrac{\Gamma \vdash \neg\neg A}{\Gamma \vdash A}\ \neg\neg_e$ est admissible en logique classique.

2.2 Du typage pour faire des preuves

La correspondance de Curry-Howard fait le lien entre les fonctions typables avec une restriction du langage OCaml et les théorèmes en logique minimale.

Par exemple, la formule de la question 22 peut se représenter avec la fonction :

```
# let f x y = y x;;
val f : 'a -> ('a -> 'b) -> 'b = <fun>
```

Réciproquement, on peut créer une fonction :

```
# let g x y = x y;;
val g : ('a -> 'b) -> 'a -> 'b = <fun>
```

Donc on peut en déduire que $(A \to B) \to A \to B$ est un théorème.

On considère les types de base suivants :

```
type bot = |

type ('a, 'b) et = 'a * 'b

type ('a, 'b) ou = G of 'a | D of 'b
```

On fait correspondre ces types à des formules propositionnelles :

- un type `'a` correspond à une formule A générique ;
- le type `bot` est un type tel qu'aucun objet de ce type ne peut exister (un type somme, sans choix possible). Il correspond à $\bot$ (la formule toujours fausse) ;
- le type `('a, 'b) et` correspond à la conjonction de `'a` et `'b`, c'est-à-dire une formule $A \wedge B$;
- le type `('a, 'b) ou` correspond à la disjonction de `'a` et `'b`, c'est-à-dire une formule $A \vee B$.

Par définition, une formule $\neg A$ correspondrait à un type `'a -> bot`.

```
type 'a non = 'a -> bot
```

On admet le résultat suivant : une formule est un théorème en logique minimale si et seulement si elle correspond au type d'une fonction OCaml écrite avec les opérations suivantes :

- abstraction : à partir d'une variable `x` et d'une expression `e`, création d'une fonction qui à `x` renvoie `e` : `fun x -> e` ;
- application : à partir d'une variable `x` et d'une expression `e`, évaluation de la fonction `x` en `e` : `x e` ;
- renommage d'une expression `e` en une variable `x` en utilisant un `let` : `let x = e in` ;
- pour les conjonctions : accès à la première et deuxième composantes ; création d'un couple ;
- pour les disjonctions : filtrage par motif `G` ou `D` ; création avec le constructeur `G` ou `D`.

En particulier, on interdit la récursivité, les boucles, les exceptions, les références, les types natifs (`int`, `bool`, ...).

2.2.1 Vérification des règles d'inférence

On peut traduire chaque règle d'inférence en un théorème, quitte à ignorer le contexte. Par exemple, la première règle $\wedge_e$ peut être vue comme le théorème $(A \wedge B) \to A$.

Lorsqu'une règle possède plusieurs prémisses, on peut les séparer par des implications. Par exemple, la règle $\wedge_i$ peut être vue comme $A \to B \to (A \wedge B)$.

Lorsqu'un séquent contient une formule dans son contexte, on peut faire passer cette formule dans la conclusion, en utilisant l'introduction de l'implication. On remarque qu'avec cette vision des choses, les règles d'introduction et d'élimination de l'implication sont en fait identiques et correspondent dans les deux cas à la formule $(A \to B) \to A \to B$.

Question 24 Écrire une fonction `et_elim : ('a, 'b) et -> 'a`, correspondant à une règle $\wedge_e$.

Question 25 Montrer qu'il existe une fonction de type `('a, 'b) ou -> ('a -> 'c) -> ('b -> 'c) -> 'c`. À quelle règle d'inférence de la logique minimale ce type correspond-il ?

2.2.2 Quelques preuves en logique minimale

Question 26 Écrire une fonction de type `('a -> 'a -> 'b) -> 'a -> 'b`. En déduire que $(A \to \neg A) \to \neg A$ est prouvable en logique minimale.

Certaines fonctions nécessitent de l'annotation de type pour avoir le bon type. Voici par exemple comment écrire une fonction dont le type correspond à la règle $\neg_e$:

```
# let non_elim ((a, non_a) : ('a, 'a non) et) = non_a a;;
val non_elim : ('a, 'a non) et -> bot = <fun>
```

Question 27 En écrivant des fonctions du bon type, montrer que $\vdash_m (\neg A \wedge \neg B) \to \neg(A \vee B)$ et que $\vdash_m \neg(A \vee B) \to (\neg A \wedge \neg B)$.

Donner un arbre de preuve du séquent $\vdash (\neg A \wedge \neg B) \to \neg(A \vee B)$ en logique minimale.

2.2.3 Logique intuitionniste et logique classique

Si on utilise la levée d'exception ou la récursivité, on peut créer des fonctions dont la signature correspond à des formules non prouvables en logique minimale. On considère les deux fonctions suivantes :

```
val bot_elim : bot -> 'a = <fun>

val tiers_exclu : 'b -> ('a, 'a non) ou = <fun>
```

Pour la suite, en plus des opérations déjà autorisées précédemment, on s'autorise l'utilisation de ces deux fonctions : `bot_elim` lorsqu'on travaille en logique intuitionniste, et `tiers_exclu` en plus de `bot_elim` en logique classique.

Question 28 En écrivant une fonction du bon type, montrer que $\vdash_i \neg(A \to B) \to B \to A$.

Question 29 En écrivant une fonction du bon type, montrer que $\vdash_c \neg\neg A \to A$.

2.3 Logique et typage

On s'intéresse maintenant au typage des expressions dans un sous-ensemble du langage OCaml. On note $\vdash e : \tau$ pour dire que l'expression e admet τ comme type (on parle alors de **jugement de typage**). On a donc par exemple $\vdash$ `123` : `int` et $\vdash$ `1 :: [3; 4]` : `int list`.

Question 30 Quel jugement de typage obtiendrait-on pour l'expression `List.map (fun x -> x + 1)` ?

Pour pouvoir typer des expressions contenant des variables libres, il est nécessaire d'avoir un contexte donnant le type de chacune de ces variables. On étend donc la notion de jugement pour inclure un contexte Γ, qui est un ensemble de jugements de la forme $x : \tau$ où les x sont des **variables distinctes** et τ un type. Par exemple :

$$\{x : \texttt{int}, l : \texttt{int list}, f : \texttt{int -> int}\} \vdash \texttt{((f x) :: l)} : \texttt{int list}$$

On définit alors un système d'inférence pour les jugements de typage, constitué de :

– pour chaque constante c du langage de type τ, on dispose d'un axiome :

$$\frac{}{\Gamma \vdash c : \tau} \text{ (type)}$$

par exemple, $\vdash$ `true` : `bool` et $\vdash$ `2.14` : `float` sont des axiomes ;
– une autre règle d'axiome (var), pour « variable »

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \text{ (var)}$$

– une règle (abs), pour « abstraction » :

$$\frac{\Gamma, x : \sigma \vdash e : \tau}{\Gamma \vdash (\texttt{fun}\ x \to e) : \sigma \to \tau} \text{ (abs)}$$

– une règle (app), pour « application » :

$$\frac{\Gamma \vdash f : \sigma \to \tau \qquad \Gamma \vdash e : \sigma}{\Gamma \vdash f\ e : \tau} \text{ (app)}$$

Question 31 On considère l'environnement de typage $\Gamma = \{f : \alpha \to (\beta \to \gamma), g : \beta \to \alpha\}$. Dériver le jugement suivant : $\Gamma, x : \beta \vdash g\ x : \alpha$.

Question 32 En déduire que, pour le même Γ, on a $\Gamma \vdash (\texttt{fun}\ x \to f(g\ x)\ x) : \beta \to \gamma$.

On souhaite à présent prouver que l'expression $e = (\texttt{fun}\ h \to h\ 1\ 2)(\texttt{fun}\ x \to 3)$ n'est pas typable, c'est-à-dire qu'il n'existe pas de contexte Γ et de type τ tels que $\Gamma \vdash e : \tau$. On procède par l'absurde et on suppose qu'il existe un tel contexte.

Question 33 Justifier qu'on a nécessairement $\Gamma \vdash (\texttt{fun}\ h \to h\ 1\ 2) : \alpha \to \tau$ et $\Gamma \vdash (\texttt{fun}\ x \to 3) : \alpha$ pour un certain type α.

Question 34 En considérant la dernière étape de la dérivation de $\Gamma \vdash (\texttt{fun}\ x \to 3) : \alpha$, dire quelle forme doit avoir le type α.

Question 35 En s'intéressant au jugement $\Gamma \vdash (\texttt{fun}\ h \to h\ 1\ 2) : \alpha \to \tau$, conclure (on admettra que `int` ne peut pas se mettre sous la forme $\gamma \to \delta$).

⋆ ⋆ ⋆

Annexe : règles d'inférence

La logique **minimale** est formée des règles suivantes :

- introduction et élimination de $\wedge$:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge_i, \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge_e \quad \text{et} \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge_e$$

- introduction et élimination de $\vee$:

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee_i, \quad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee_i \quad \text{et} \quad \frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee_e$$

- introduction et élimination de $\to$:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to_i \quad \text{et} \quad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \to_e$$

- axiome et affaiblissement :

$$\frac{}{\Gamma, A \vdash A}\ (\text{ax}) \quad \text{et} \quad \frac{\Gamma \vdash B}{\Delta \vdash B}\ (\text{aff}) \text{ pour } \Gamma \subseteq \Delta$$

La logique **intuitionniste** est formée des règles de la logique minimale et de l'élimination de $\bot$:

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \bot_e$$

La logique **classique** est formée des règles de la logique intuitionniste et du tiers-exclu :

$$\frac{}{\Gamma \vdash A \vee \neg A}\ (\text{te})$$