Consignes générales

L'examen dure deux heures. Tous les documents sont autorisés. Tout matériel électronique doté d'une fonction de communication doit être silencieux et rangé hors de vue.

1 Introduction

Le but de cet examen est d'étudier la construction de filtrage de motifs présente en Hopix, OCaml, Haskell, Scala, Kotlin, etc. Chacun des trois exercices concerne l'une des facettes de cette construction : sa sémantique, sa compilation naïve, et sa compilation optimisée. Le sujet est **volontairement trop long** : il suffit d'avoir fait deux exercices parfaitement pour avoir 20/20. Les exercices sont indépendants les uns des autres, mais il est fortement recommandé d'avoir lu l'intégralité du sujet avant de commencer à répondre.

Le filtrage de motifs permet de discriminer entre les *valeurs* du langage selon leur forme, celle-ci étant spécifiée par un *motif*. Dans cet examen, on va s'intéresser à une construction de filtrage réduite à sa plus simple expression. Les seules valeurs considérées sont les valeurs des types sommes, étiquettées par un constructeur K. Les seuls motifs considérés sont le motif attrape-tout (ou *wildcard* en anglais), les motifs OU, et les motifs étiquettés.

$$v ::= K\langle v, \dots, v\rangle \qquad\qquad p ::= \_ \mid (p \,||\, p) \mid K\langle p, \dots, p\rangle$$

On peut voir une valeur comme un arbre n-aire dont les nœuds sont étiquettés par un constructeur K ; par exemple, $\mathsf{Nil}\langle\rangle$, $\mathsf{Unit}\langle\rangle$ et $\mathsf{Cons}\langle\mathsf{Unit}\langle\rangle, \mathsf{Cons}\langle\mathsf{Unit}\langle\rangle, \mathsf{Nil}\langle\rangle\rangle\rangle$ sont des valeurs. On suppose que chaque constructeur a une *arité* $r \in \mathbb{N}$, qui fixe le nombre de sous-arbres autorisé ; par exemple, Unit et Nil sont d'arité 0 tandis que Cons est d'arité 2. Les motifs sont eux aussi très simplifiés, puisque dépourvus de variables. Cette restriction permet de définir très simplement sous quelles conditions un motif p *filtre* (accepte) une valeur v, auquel cas on écrit $p \preceq v$.

$$\boxed{p \preceq v} \qquad \frac{}{\_ \preceq v} \qquad \frac{p_1 \preceq v}{p_1 \,||\, p_2 \preceq v} \qquad \frac{p_2 \preceq v}{p_1 \,||\, p_2 \preceq v} \qquad \frac{p_1 \preceq v_1 \qquad \dots \qquad p_n \preceq v_n}{K\langle p_1, \dots, p_n\rangle \preceq K\langle v_1, \dots, v_n\rangle}$$

Par extension, on considère qu'un vecteur (liste) de motifs $\vec{p}$ filtre un vecteur de valeurs $\vec{v}$, ce qui est noté $\vec{p} \preceq \vec{v}$, lorsque les deux vecteurs sont de même longeur n et que $\vec{p}_i \preceq \vec{v}_i$ pour tout $i \in [1, n]$.

Enfin, on suppose que la construction de filtrage de motif s'applique à des vecteurs de valeurs, qu'elle filtre par des vecteurs de motifs. Par souci de simplicité, on va supposer que le résultat renvoyé par chaque branche est un simple entier, ce qui nous évite d'avoir à spécifier le reste du langage. La construction de filtrage prend alors la forme $\mathsf{match}\ \vec{v}\ \mathsf{with}\ P \Rightarrow A$, où $\vec{v}$ désigne un vecteur de valeurs, P une matrice de motifs, et A un vecteur colonne d'autant d'entiers que P contient de lignes.

$$\mathsf{match}\ \underbrace{\begin{pmatrix} v_1 & \dots & v_n \end{pmatrix}}_{\vec{v}}\ \mathsf{with}\ \underbrace{\begin{pmatrix} p_1^1 & \dots & p_n^1 \\ \vdots & & \vdots \\ p_1^m & \dots & p_n^m \end{pmatrix}}_{P} \Rightarrow \underbrace{\begin{pmatrix} a^1 \\ \vdots \\ a^m \end{pmatrix}}_{A}$$

Le résultat de l'évaluation de ce filtrage est l'entier a^j si le $j$ème vecteur ligne de P est le premier à filtrer le vecteur $\vec{v}$. Plus formellement, l'évaluation renvoie a^j lorsque j est le plus petit entier tel que $\begin{pmatrix} p_1^j & \dots & p_n^j \end{pmatrix} \preceq \vec{v}$. La contrainte de minimalité sur j reflète la sémantique standard selon laquelle les lignes sont testées successivement de haut en bas. On écrira alors $\mathcal{M}(\vec{v}, P \Rightarrow A) = a^j$ pour signifier que le filtrage fonctionne et résulte en l'entier a^j.

2 Sémantique et interprétation (10 points)

Dans cet exercice, on se propose de rédiger en OCaml un interprète naïf pour le filtrage de motifs tel qu'exposé ci-dessus. Le type des valeurs est défini ci-dessous. Le type des constructeurs est volontairement laissé abstrait.

```
type valeur = C of constructeur * valeur list
```

1. (2 points) Pour chacune des paires (p_i, v_i) ci-dessous, dire si $p_i \preceq v_i$. Vous n'avez pas à justifier vos réponses.

$$\_ \preceq_? \mathsf{Unit}\langle\rangle \qquad \mathsf{Unit}\langle\_\rangle \preceq_? \mathsf{Unit}\langle\rangle \qquad \mathsf{A}\langle\_, \mathsf{B}\langle\rangle\rangle \preceq_? \mathsf{A}\langle\mathsf{C}\langle\rangle, \mathsf{B}\langle\rangle\rangle \qquad \mathsf{A}\langle\mathsf{C}\langle\rangle, \_\rangle \preceq_? \mathsf{A}\langle\mathsf{D}\langle\rangle, \mathsf{C}\langle\rangle\rangle$$

Solution: oui, non, oui, non.

2. (4 points) Définir un type `motif` pour représenter les motifs de notre langage. Définir ensuite une fonction `filtre : motif -> valeur -> bool` telle que `filtre p v` s'évalue vers `true` ssi p $\preceq$ v.

Solution:

```
type motif = PAny
           | POr of motif * motif
           | PConstr of constructeur * motif list

let rec filtre p v = match p with
  | PAny -> true
  | POr (p, q) -> filtre p v || filtre q v
  | PConstr (k, ps) -> let C (k', vs) = v in k = k' && filtre_tous ps vs

and filtre_tous ps vs =
  List.length ps = List.length vs && List.for_all2 filtre ps vs
```

3. (4 points) Définir une fonction

```
val eval_filtrage : (motif list * int) list -> valeur list -> int option
```

telle que `eval_filtrage br vs` s'évalue soit vers `Some a` lorsque `a` est le résultat du filtrage du vecteur de valeurs `vs` par les branches `br`, soit vers `None` si ce filtrage échoue. Avec les notations de la section précédente, le $j$ème élément de `br` fournit à la fois la $j$ème ligne P et le $j$ème entier de A.

Solution:

```
let rec eval_filtrage br vs =
  match br with
  | [] -> None
  | (ps, a) :: br -> if filtre_tous ps vs then Some a
                                          else eval_filtrage br vs
```

3 Compilation (10 + 10 points)

Le but de cette deuxième section est de compiler le filtrage de motif, d'abord de façon naïve, puis de façon plus astucieuse et efficace. Les deux traductions emploient le même langage cible, qui peut être vu comme un petit sous-ensemble de Hobix.

$$\alpha ::= \mathsf{Return}(a) \mid \mathsf{Fail} \mid \mathsf{Switch}(\ell, \beta) \mid \mathsf{Read}(\ell[i], \ell.\, \alpha) \qquad \beta ::= K \mapsto \alpha; \ldots; K \mapsto \alpha; [* \mapsto \alpha]$$

Un *emplacement* ℓ représente une adresse mémoire abstraite à laquelle se trouve une valeur. Une *action* α peut être l'action $\mathsf{Return}(a)$, qui réussit systématiquement en renvoyant $a \in \mathbb{N}$, l'action Fail qui échoue systématiquement, l'action $\mathsf{Switch}(\ell, \beta)$ qui teste le constructeur de la valeur située à l'emplacement ℓ, et l'action $\mathsf{Read}(\ell[i], \ell'.\, \alpha)$ qui lie ℓ' dans l'action α à la $i$ème sous-valeur de la sous-valeur située à l'emplacement ℓ. Une liste de *branches* β spécifie un ensemble fini de constructeurs K_i (sans répétitions), noté $dom(\beta)$, ainsi que les actions α_i qui leur sont associées.

Elle peut optionnellement spécifier une action par défaut à exécuter lorsque le constructeur de la valeur testée n'appartient pas à $dom(\beta)$.

Pour éviter d'éventuelles ambiguités, on va doter le langage des actions d'une sémantique. Elle prend la forme d'un jugement inductif $E \vdash \alpha \rightsquigarrow a$, qui indique que l'action α s'évalue vers l'entier a dans l'environnement E. Un environnement est une fonction partielle finie des emplacements dans les valeurs ; la notation $E(\ell)$ suppose que ℓ appartienne bien au domaine de E. Les règles qui définissent le jugement sont données ci-dessous.

$$\boxed{E \vdash \alpha \rightsquigarrow a} \qquad \frac{}{E \vdash \mathsf{Return}(a) \rightsquigarrow a} \qquad \frac{E(\ell) = K_i\langle \ldots \rangle \qquad (K_i \mapsto \alpha_i) \in \beta \qquad E \vdash \alpha_i \rightsquigarrow a}{E \vdash \mathsf{Switch}(\ell, \beta) \rightsquigarrow a}$$

$$\frac{E(\ell) = K\langle \ldots \rangle \qquad K \notin dom(\beta) \qquad (* \mapsto \alpha) \in \beta \qquad E \vdash \alpha \rightsquigarrow a}{E \vdash \mathsf{Switch}(\ell, \beta) \rightsquigarrow a} \qquad \frac{E(\ell) = K\langle v_1, \ldots, v_n \rangle \qquad i \in [1, n] \qquad E[\ell' \mapsto v_i] \vdash \alpha \rightsquigarrow a}{E \vdash \mathsf{Read}(\ell[i], \ell'.\, \alpha) \rightsquigarrow a}$$

Notons que cette sémantique est partielle : pour certaines actions α et environnements E, il n'existe aucun entier a tel que $E \vdash \alpha \rightsquigarrow a$. C'est le cas par exemple pour l'action Fail dans tous les cas, ou bien pour $\mathsf{Switch}(\ell, \beta)$ lorsque la valeur située à l'emplacement ℓ n'appartient pas à $dom(\beta)$ et que β ne spécifie pas de cas par défaut.

Le langage des actions étant maintenant défini, nous pouvons spécifier ce qu'est une fonction de compilation correcte. Une telle fonction $\mathcal{F}$ reçoit un vecteur d'emplacements $\vec{\ell}$ et une matrice de filtrage $P \Rightarrow A$, et renvoie une action qui filtre les valeurs situées aux emplacements $\vec{\ell}$ dans l'environnement. Une telle fonction est correcte lorsque

$$\mathcal{M}\left(\begin{pmatrix} v_1 & \ldots & v_n \end{pmatrix}, P \Rightarrow A\right) = a \quad \Leftrightarrow \quad [\ell_i \mapsto v_i]_{i \in [1,n]} \vdash \mathcal{F}\left(\begin{pmatrix} \ell_1 & \ldots & \ell_n \end{pmatrix}, P \Rightarrow A\right) \rightsquigarrow a.$$

3.1 Compilation ligne-à-ligne (10 points)

Le but de cet exercice est de compiler le filtrage d'une façon inefficace mais simple à comprendre et implémenter. On va pour ce faire définir une fonction de compilation $\mathcal{R}$ qui traverse la matrice de filtrage P ligne-à-ligne, en suivant directement la sémantique du filtrage.

1. (7 points) L'essentiel de la compilation est effectué par deux fonctions mutuellement récursives $\mathcal{R}_{pat}$ et $\mathcal{R}_{vec}$.
   - La fonction $\mathcal{R}_{pat}$ reçoit un emplacement ℓ, un motif p, une action α_s et une action α_f. Elle renvoie une action qui s'évalue soit vers le résultat de α_s lorsque la valeur située à l'emplacement ℓ est filtrée par p, soit vers le résultat α_f sinon.
   - La fonction $\mathcal{R}_{vec}$ reçoit un vecteur d'emplacements $\vec{\ell}$, un vecteur de motifs $\vec{p}$, une action α_s et une action α_f. Elle renvoie une action qui s'évalue soit vers le résultat de α_s lorsque les valeurs situées aux emplacements $\vec{\ell}$ sont filtrées par les motifs $\vec{p}$, soit vers le résultat α_f sinon.

   Compléter le pseudo-code ci-dessous. Les deux vecteurs passés à $\mathcal{R}_{vec}$ sont supposés de même longueur.

$$\begin{array}{l} \mathcal{R}_{pat}(\ell, \quad \_ \quad, \alpha_s, \alpha_f) = ? \\ \mathcal{R}_{pat}(\ell, \quad p_1 \,||\, p_2 \quad, \alpha_s, \alpha_f) = ? \\ \mathcal{R}_{pat}(\ell, K\langle p_1, \ldots, p_n \rangle, \alpha_s, \alpha_f) = ? \end{array} \qquad \begin{array}{l} \mathcal{R}_{vec}(\ () \ , \ () \ , \alpha_s, \alpha_f) = ? \\ \mathcal{R}_{vec}((\ell\, \vec{\ell}),(p\, \vec{p}), \alpha_s, \alpha_f) = ? \end{array}$$

Solution:

$$\begin{array}{rl} \mathcal{R}_{pat}(\ell, \ \_, \alpha_s, \alpha_f) &= \alpha_s \\ \mathcal{R}_{pat}(\ell, p_1 \,||\, p_2, \alpha_s, \alpha_f) &= \mathcal{R}_{pat}(\ell, p_1, \alpha_s, \mathcal{R}_{pat}(\ell, p_2, \alpha_s, \alpha_f)) \\ \mathcal{R}_{pat}(\ell, K\langle \vec{p} \rangle, \alpha_s, \alpha_f) &= \mathsf{Switch}(\ell, [K \mapsto \vec{\mathsf{Read}}(\ell, \vec{\ell'}.\, \mathcal{R}_{vec}(\vec{\ell'}, \vec{p}, \alpha_s, \alpha_f)); * \mapsto \alpha_f]) \\ & \quad \text{où } \vec{\ell'} \text{ est un vecteur d'emplacements frais de même taille que } \vec{p} \\ & \quad \text{et } \vec{\mathsf{Read}}(\ell, (\ell'_1, \ldots, \ell'_n).\, \alpha) \triangleq \mathsf{Read}(\ell[1], \ell'_1. \ldots \ \mathsf{Read}(\ell[n], \ell'_n.\, \alpha)) \\ \mathcal{R}_{vec}((), (), \alpha_s, \alpha_f) &= \alpha_f \\ \mathcal{R}_{vec}(\ell\, \vec{\ell}, p\, \vec{p}, \alpha_s, \alpha_f) &= \mathcal{R}_{pat}(\ell, p, \mathcal{R}_{vec}(\vec{\ell}, \vec{p}, \alpha_s, \alpha_f), \alpha_f) \end{array}$$

2. (3 points) On peut définir la fonction de compilation complète $\mathcal{R}$ en utilisant les fonctions précédentes. Il s'agit d'une fonction récursive dont le cas de base est celui de la matrice vide (qui n'a aucune ligne). Donner, dans le même style que les définitions précédentes, le pseudo-code de $\mathcal{R}(\vec{\ell}, P \Rightarrow A)$.

Solution:

$$\mathcal{R}(\vec{\ell}, (\,) \Rightarrow (\,)) = \mathsf{Fail}$$
$$\mathcal{R}(\vec{\ell}, (\vec{p}\,P) \Rightarrow (a\,\vec{A})) = \mathcal{R}_{vec}(\vec{\ell}, \vec{p}, \mathsf{Return}(a), \mathcal{R}(\vec{\ell}, P \Rightarrow \vec{A}))$$

3.2 Compilation vers les arbres de décision (10 points)

La compilation ligne-à-ligne produit du code peu efficace, dans la mesure où le constructeur d'une valeur étiquetté va potentiellement être testé (et lu en mémoire) de nombreuses fois. Une alternative est de construire un *arbre de décision*, où chaque constructeur n'est examiné qu'une seule fois. On peut comprendre cet algorithme comme parcourant la matrice colonne par colonne plutôt que ligne par ligne. L'algorithme $\mathcal{C}(\vec{\ell}, P \Rightarrow A)$ est le suivant.

1. Si P est vide, $\mathcal{C}$ renvoie Fail.
2. Si P commence par une ligne formée uniquement de motifs attrape-tout, $\mathcal{C}$ renvoie $\mathsf{Return}(a^1)$ puisque tout vecteur de valeurs va être filtré par cette ligne.
3. Sinon, au moins une colonne de la matrice contient un motif de la forme $K\langle p_1, \ldots, p_n\rangle$. On choisit l'une de ces colonnes, qu'on appelle i. Soit Σ l'ensemble des constructeurs qui apparaîssent en tête des motifs de la colonne i. Alors $\mathcal{C}$ renvoie $\mathsf{Switch}(\ell_i, \beta)$, où β est construit à partir de Σ et d'appels récursifs à $\mathcal{C}$.

Le but de cet exercice est de construire β, dans le cas simple où Σ contient tous les constructeurs possibles, c'est-à-dire où le filtrage de la $i$ème valeur est exhaustif.

1. (5 points) Pour décrire les appels récursifs, on a besoin de *spécialiser* la matrice sur un constructeur. La matrice spécialisée $\mathcal{S}(i, K, P)$ décrit le filtrage qu'il reste à réaliser par P après avoir filtré le constructeur K sur la $i$ème valeur d'entrée. La spécialisation de P tout entière est obtenue en spécialisant indépendamment chaque ligne. Compléter le pseudo-code ci-dessous, qui donne la spécialisation d'une ligne. *Indication* : selon le cas, la spécialisation d'une ligne peut produire 0, 1 ou 2 nouvelles lignes.

$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad K\langle q_1, \ldots, q_r\rangle \quad p_{i+1} \quad \ldots \quad p_n)) = ?$$
$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad K'\langle q_1, \ldots, q_{r'}\rangle \quad p_{i+1} \quad \ldots \quad p_n)) = ? \qquad (K \neq K')$$
$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad \_ \quad p_{i+1} \quad \ldots \quad p_n)) = ?$$
$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad p_1 \,||\, p'_1 \quad p_{i+1} \quad \ldots \quad p_n)) = ?$$

Solution:

$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad K\langle q_1, \ldots, q_r\rangle \quad p_{i+1} \quad \ldots \quad p_n)) = (p_1 \quad \ldots \quad p_{i-1} \quad q_1 \quad \ldots \quad q_r \quad p_{i+1} \quad \ldots \quad p_n)$$
$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad K'\langle q_1, \ldots, q_{r'}\rangle \quad p_{i+1} \quad \ldots \quad p_n)) = \text{la matrice vide} \qquad (K \neq K')$$
$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad \_ \quad p_{i+1} \quad \ldots \quad p_n)) = (p_1 \quad \ldots \quad p_{i-1} \quad \_ \quad \ldots \quad \_ \quad p_2 \quad \ldots \quad p_n)$$
$$\mathcal{S}(i, K, (p_1 \quad \ldots \quad p_{i-1} \quad p_1 \,||\, p'_1 \quad p_{i+1} \quad \ldots \quad p_n)) = \begin{pmatrix} \mathcal{S}(i, K, (p_1 \quad p_2 \quad \ldots \quad p_n)) \\ \mathcal{S}(i, K, (p'_1 \quad p_2 \quad \ldots \quad p_n)) \end{pmatrix}$$

2. (5 points) Supposons que le constructeur $K_j \in \Sigma$ soit d'arité r. Donner l'action α_j associée à K_j dans β. *Indication* : l'appel récursif à $\mathcal{C}$ doit utiliser la fonction $\mathcal{S}$ définie à la question précédente.

Solution:

$$\beta_j = \vec{\mathsf{Read}}(\ell, \vec{\ell'}.\, \mathcal{C}(\vec{\ell}, \mathcal{S}(i, K_j, P) \Rightarrow A'))$$

où, $\vec{\ell'}$ est un vecteur frais de largeur r et $\vec{\mathsf{Read}}(\ell, \vec{\ell'}.\, \alpha)$ est défini de la même façon qu'à la question précédente. La matrice A' est obtenue en dupliquant, préservant ou copiant les lignes de A en accord avec les transformations effectuées par $\mathcal{S}(i, K_j, P)$.