Université de Marne-la-Vallée
Licence d'informatique et IUP GMI
Examen de programmation fonctionnelle. Janvier 2002
Durée 2 heures. Tous les documents sont autorisés.

Tous les exercices sont indépendants. N'oubliez pas de tourner la page. Dans les exercices 1 et 2, les programmes sont à écrire en Objective Caml. Un corrigé sera sur le web après l'examen.

Exercice 1.— (13 points)

1.1) (1 point)

Écrire une fonction `pair` qui prend en argument un entier et renvoie `true` si l'entier est pair et `false` sinon.

1.2) (2 points)

Écrire une fonction `exists` qui prend en argument un prédicat `p` (i.e. une fonction `'a -> bool`) et une liste d'éléments auxquels peut s'appliquer `p` et qui teste s'il y a au moins un élément qui satisfait le prédicat `p` dans la liste. On aura ainsi

```
# exists pair [1;3;5];;
 - : bool = false
# exists pair [1;3;4;5];;
 - : bool = true
```

Quel est le type de `exists` ?

1.3) (2 points)

Ré-écrire la fonction `exists` en utilisant `List.map` et `List.fold_right`. On rappelle ci-dessous leurs types.

```
# List.map;;
- : f:('a -> 'b) -> 'a list -> 'b list = <fun>
# List.fold_right;;
- : f:('a -> 'b -> 'b) -> 'a list -> init:'b -> 'b = <fun>
```

1.4) (2 points)

Un arbre planaire général est un arbre dont on ne fixe pas a priori le nombre de fils : à chaque noeud on associe une liste de fils dont la longueur peut varier. On considère dans la suite de l'exercice de tels arbres dont les noeuds ont une étiquette de type `'a`. On utilisera pour les représenter le type suivant, où `Vide` représente l'arbre vide.

```
# type 'a arbre =
           Vide
         | Noeud of 'a * 'a arbre list ;;
```

Écrire une fonction `hauteur` qui calcule la hauteur d'un arbre. La hauteur de l'arbre vide est 0. On pourra utiliser la fonction `max`.

```
#  max 2 3;;
- : int = 3
```

1.5) (2 points)

Écrire une fonction `map_arbre` qui prend en argument une fonction `f` et un arbre `t` et renvoie l'arbre obtenu à partir de `t` en appliquant la fonction `f` à chaque étiquette d'un noeud. Quel est le type de `map_arbre` ?

1.6) (2 points)

Écrire une fonction `exists_arbre` qui prend en argument un prédicat `p` et un arbre `t` tel que `p` peut s'appliquer aux étiquettes de `t`, et qui teste s'il y a au moins une étiquette de `t` qui satisfait le prédicat `p`. Donner le type de `exists_arbre`.

1.7) (2 points)

À l'aide de `exists_arbre`, écrire une fonction `element` qui prend un argument `x` et un arbre `t`, et renvoie `true` si `x` est une étiquette de `t` et `false` sinon. Donner le type de `element`.

Exercice 2.— (4 points)

2.1) (2 points)

Écrire une fonction `fusion_i` qui prend en entrée deux listes d'entiers triées en ordre croissant et retourne une nouvelle liste triée contenant les éléments des deux premières.

2.2) (2 points)

Écrire une fonction `fusion` générale, qui prend en argument une fonction de comparaison et deux listes triées selon cet ordre et retourne la liste fusionnée selon le même ordre. La fonction de comparaison sera du type `'a -> 'a -> bool`. On aura ainsi

```
# let comp x y = x<y;;
```

```
 val c : 'a -> 'a -> bool = <fun>
# fusion comp [1;3;7] [2;4;8];;
- : int list = [1; 2; 3; 4; 7; 8]
```

Exercice 3.— (3 points)
Soient a, b, c, d des atomes (ou variables) du calcul des propositions.
3.1) (2 points)
Donner une preuve syntaxique de la formule suivante. Le système de déduction à utiliser est le système de Hilbert présenté en cours. On pourra utiliser tous les résultats du cours (notamment le théorème de déduction), à l'exclusion toutefois du théorème de complétude.
$(a \to (b \to (c \to d))) \to ((a \to b) \to (a \to (c \to d)))$;
3.2) (1 point)
Définir une fonction Objective Caml de type
`('a->'b->'c->'d)->('a->'b)->'a->'c->'d`

Exercice 4.— (3 points)
On considère des formules du calcul propositionnel où les atomes propositionnels sont ordonnés. On dit qu'une formule F d'arité n (i.e. comportant n variables distinctes) est de Sheffer si pour toute formule G il existe une formule, bâtie à partir de F, des variables de proposition et de l'opérateur de substitution, qui est équivalente à G.
Une telle formule F définit une fonction $\tilde{F} : B^n \to B$ où B désigne l'ensemble $\{vrai, faux\}$ des valeurs de vérité.
Si une formule d'arité n est de Sheffer, on peut la voir comme un connecteur complet d'arité n.
4.1) (1 point)
Montrer que la formule $F = \neg(p \wedge q)$, d'arité 2, où p, q sont des atomes propositionnels, est de Sheffer.
4.2) (2 points)
Montrer que si F est de Sheffer, alors les trois conditions suivantes sont vérifiées.
- $\tilde{F}(vrai, \ldots, vrai) = faux$
- $\tilde{F}(faux, \ldots, faux) = vrai$
- Il existe $(\alpha_1, \ldots, \alpha_n)$ dans B^n tel que $\tilde{F}(\alpha_1, \ldots, \alpha_n) = \tilde{F}(\neg\alpha_1, \ldots, \neg\alpha_n)$