Université Paris 6
Licence d'informatique

Module de Programmation
ANNÉE 1999-2000

Contrôle Continu - 3 décembre 1999
CORRECTION

Exercice 1

Un permis de conduire français contient, entre autres informations, l'identité de son possesseur et la liste des catégories de permis de cette personne.

Les catégories de permis considérés ici sont `A` (permis moto) et `B` (voiture).

Une personne est identifiée par son nom et sa ville donnée par son code postal.

1 Définir les types `catégorie_permis`, `identité` et `permis`.

```
#type catégorie_permis = A | B ;;
Le type catégorie_permis est défini.

#type identité = { nom: string; ville: int };;
Le type identité est défini.

#type permis = { personne: identité; catégories: catégorie_permis list };;
Le type permis est défini.
```

Par la suite, nous considérerons qu'une personne qui n'a pas de permis de conduire dispose d'un élément de type `permis`, mais que la liste des catégories de permis qu'il possède est vide.

2 Écrire la fonction `permis_voiture` qui détermine si la personne détentrice d'un permis `p` peut conduire une voiture.

Écrire la fonction `nouveau_permis` qui met à jour le permis `p` d'une personne qui vient d'être reçue à l'examen du permis de conduire pour la catégorie `c`.

```
#let permis_voiture p = mem B p.catégories;;
permis_voiture : permis -> bool = <fun>

#let nouveau_permis p c =
#  { personne = p.personne; catégories = c::p.catégories };;
nouveau_permis : permis -> catégorie_permis -> permis = <fun>
```

3 Sachant que le département se déduit du code postal en divisant ce dernier par 1000, écrire la fonction `département_of_ville` qui indique le département associé à la ville `v`.

```
#let département_of_ville v = v / 1000;;
département_of_ville : int -> int = <fun>
```

4 À la suite d'un contrôle routier, un policier a relevé une liste de permis de contrevenants, il souhaite classer ces permis selon l'ordre alphabétique des noms de ces personnes. Écrire la fonction `tri_alph_nom` qui trie une liste de permis selon l'ordre alphabétique des noms de leurs détenteurs. Vous pourrez pour cela utiliser la fonction `tri_insertion` vue en cours ou bien la fonction `sort` du module `sort` qui fait partie de la bibliothèque standard de Caml.

```
#let ordre_alph_nom p1 p2 = p1.personne.nom < p2.personne.nom;;
ordre_alph_nom : permis -> permis -> bool = <fun>

#
#let tri_alph_nom pl = sort__sort ordre_alph_nom pl;;
tri_alph_nom : permis list -> permis list = <fun>
```

5 Devant la variété d'origine de ces détenteurs de permis, ce policier décide de les classer d'abord par ordre de département et pour chaque département en ordonnant les noms des contrevenants. Écrire la fonction `tri_plus_complexe` associée.

```
#let ordre_plus_complexe p1 p2 =
#  let d1 = département_of_ville p1.personne.ville
#  and d2 = département_of_ville p2.personne.ville in
#    d1 < d2 || d1 = d2 && p1.personne.nom < p2.personne.nom;;
ordre_plus_complexe : permis -> permis -> bool = <fun>

#
#let tri_plus_complexe pl = sort__sort ordre_plus_complexe pl;;
tri_plus_complexe : permis list -> permis list = <fun>
```

6 Écrire une fonction `détenteurs_permis_voiture` qui extrait d'une liste de permis ceux qui permettent de conduire une voiture. On écrira cette fonction de deux façons différentes, tout d'abord directement récursivement, puis en utilisant l'itérateur `filter` vu en cours.

```
#let rec détenteurs_permis_voiture pl = match pl with
#  | [] -> []
#  | p::pl' -> if permis_voiture p
#              then p::(détenteurs_permis_voiture pl')
#              else détenteurs_permis_voiture pl';;
détenteurs_permis_voiture : permis list -> permis list = <fun>

#(* ou *)
#(* Rappel: *)
#let rec filter p l = match l with
#| [] -> []
#| e::l' -> if p e then e::(filter p l') else filter p l';;
filter : ('a -> bool) -> 'a list -> 'a list = <fun>

#(* *)
#let détenteurs_permis_voiture pl = filter permis_voiture pl;;
détenteurs_permis_voiture : permis list -> permis list = <fun>
```

Exercice 2

La session débute dans l'environnement de typage global `E0`. Les environnements seront ensuite nommés `E1`, `E2`,...

1

```
#let x = 1;;
x : int = 1
```

$$E_0 \vdash 1 : \texttt{int}$$

$E_1 = \lfloor (x, \texttt{int}) \triangleleft\ E_0 \rfloor$.

2 `#let f1 = function y -> y + x;;`

```
f1 : int -> int = <fun>
```

$$\cfrac{\cfrac{E_1' \vdash y : \tau_y \quad E_1' \vdash x : \texttt{int}}{\lfloor (y, \tau_y) \triangleleft\ E_1 \rfloor \vdash y + x : \texttt{int}} \quad \boxed{\tau_y = \texttt{int}}}{E_1 \vdash \texttt{fun}\ \ y \to y + x : \texttt{int} \to \texttt{int}}$$

$E_2 = \lfloor (f1, \texttt{int} \to \texttt{int}) \triangleleft\ E_1 \rfloor$.

3

```
#let f2 =
#  function x -> function y ->
#    if x > y then x+y else f1(x-y);;
f2 : int -> int -> int = <fun>
```

$$\dfrac{\dfrac{E_2' \vdash x : \tau_x \;\; E_2' \vdash y : \tau_y}{E_2' \vdash x > y : bool} \quad \boxed{\tau_x = \tau_y} \qquad \dfrac{E_2' \vdash x : \tau_x \;\; E_2' \vdash y : \tau_x}{E_2' \vdash x + y : \texttt{int}} \quad \boxed{\tau_x = \texttt{int}} \quad (**)}{\dfrac{E_2' = \lfloor (y, \tau_y), (x, \tau_x) \triangleleft \; E_2 \rfloor \vdash \texttt{if} \;\; x > y \;\; \texttt{then} \;\; x + y \;\; \texttt{else} \;\; f1(x-y) : \texttt{int}}{E_2 \vdash \texttt{fun} \;\; x \; y \to \texttt{if} \;\; x > y \;\; \texttt{then} \;\; x + y \;\; \texttt{else} \;\; f1(x-y) : \texttt{int} \to \texttt{int} \to \texttt{int}}}$$

avec (**) :

$$\dfrac{E_2' \vdash f1 : \texttt{int} \to \texttt{int} \;\; E_2' \vdash x - y : \texttt{int}}{E_2' \vdash f1(x-y) : \texttt{int}}$$

$E_3 = \lfloor (f2, \texttt{int} \to \texttt{int} \to \texttt{int}) \triangleleft \; E_2 \rfloor.$

4
```
#let deriv f dx x = (f(x +. dx) -. f(x)) /. dx;;
deriv : (float -> float) -> float -> float -> float = <fun>
```

$$\dfrac{\dfrac{\dfrac{(**) \;\; (***)}{E_3' \vdash (f(x + .dx) - .f(x)) : \texttt{float}} \quad \boxed{\tau_4 = \texttt{float}} \quad E_3' \vdash dx : \texttt{float}}{E_3' = \lfloor (x, \tau_1), (dx, \tau_2) \triangleleft \; \lfloor (f, \tau_3) \triangleleft \; E_3 \rfloor \rfloor \vdash (f(x + .dx) - .f(x))/.dx : \texttt{float}}}{E_3 \vdash \texttt{fun} \;\; f \; dx \; x \to (f(x + .dx) - .f(x))/.dx : (\texttt{float} \to \texttt{float}) \to \texttt{float} \to \texttt{float} \to \texttt{float}}$$

avec (**) :

$$\dfrac{E_3' \vdash f : \tau_3 \qquad \dfrac{E_3' \vdash x : \tau_1 \quad E_3' \vdash dx : \tau_2}{E_3' \vdash x + .dx : \texttt{float}} \quad \boxed{\tau_1 = \tau_2 = \texttt{float}}}{E_3' \vdash f(x + .dx) : \tau_4} \quad \boxed{\tau_3 = \texttt{float} \to \tau_4}$$

avec (***) :

$$\dfrac{E_3' \vdash f : \texttt{float} \to \tau_4 \quad E_3' \vdash x : \texttt{float}}{E_3' \vdash f(x) : \tau_4}$$

$E_4 = \lfloor ((\texttt{float} \to \texttt{float}) \to \texttt{float} \to \texttt{float} \to \texttt{float}) \triangleleft \; E_3 \rfloor.$

5
```
#let partielle = deriv (function x -> x *. x) 0.001;;
partielle : float -> float = <fun>
```

$$\dfrac{(**) \quad E_4 \vdash 0.001 : \texttt{float}}{E_4 \vdash deriv \; (\texttt{fun} \;\; x \to x * .x) 0.001 : \texttt{float} \to \texttt{float}}$$

avec (**) :

$$\dfrac{E_4 \vdash deriv : (\texttt{float} \to \texttt{float}) \to \texttt{float} \to \texttt{float} \to \texttt{float} \quad E_4 \vdash (\texttt{fun} \;\; x \to x * .x) : \texttt{float} \to \texttt{float}}{E_4 \vdash deriv(\texttt{fun} \;\; x \to x * .x) : \texttt{float} \to \texttt{float} \to \texttt{float}}$$

$E_5 = \lfloor (partielle, \texttt{float} \to \texttt{float}) \triangleleft \; E_4 \rfloor.$

6 Donnez, sans détailler, le type de cette fonction et de son application :

```
#let rec iter f n =
#  if n = 0 then 1 else f (iter f (n-1));;
iter : (int -> int) -> int -> int = <fun>
```

```
#iter f1 1000;;
- : int = 1001
```

Exercice 3

Typez complètement chacune de ces déclarations en montrant également l'évolution de l'environnement. On supposera que `fst : 'a * 'b -> 'a` et `snd : 'a * 'b -> 'b`.

1

```
#let copie x = (x,x);;
copie : 'a -> 'a * 'a = <fun>
```

$$\cfrac{\cfrac{E'_o \vdash x : \tau_x \quad E'_o \vdash x : \tau_x}{E'_0 = \lfloor (x, \tau_x) \triangleleft\ E_0 \rfloor \vdash (x,x) : \tau_x * \tau_x}}{E_0 \vdash \texttt{fun}\ \ x \to (x,x) : \tau_x \to \tau_x * \tau_x}$$

Le type est généralisé.

$E_1 = \lfloor (copie, \forall \tau_x . \tau_x \to \tau_x * \tau_x) \triangleleft\ E_0 \rfloor$

2 `#let f (g,x) = g(snd x),g(fst x);;`

`f : ('a -> 'b) * ('a * 'a) -> 'b * 'b = <fun>`

$$\cfrac{\cfrac{(**) \quad (***)}{E'_1 = \lfloor (g, \tau_g), (x, \tau_x) \triangleleft\ E_1 \rfloor \vdash g(snd\ x), g(fst\ x) : \tau * \tau}}{E_1 \vdash \texttt{fun}\ \ (g,x) \to g(snd\ x), g(fst\ x) : (\tau_1 \to \tau) * (\tau_1 * \tau_1) \to (\tau * \tau)}$$

avec (**) :

$$\cfrac{E'_1 \vdash g : \tau_g \qquad \cfrac{E'_1 \vdash snd : \tau_1 * \tau_2 \to \tau_2 \quad E'_1 \vdash x : \tau_x}{E'_1 \vdash snd\ x : \tau_2}\ \boxed{\tau_x = \tau_1 * \tau_2}}{E'_1 \vdash g(snd\ x) : \tau}\ \boxed{\tau_g = \tau_2 \to \tau}$$

avec (***) :

$$\cfrac{E'_1 \vdash g : \tau_2 \to \tau \qquad \cfrac{E'_1 \vdash fst : \tau_3 * \tau_4 \to \tau_3 \quad E'_1 \vdash x : \tau_1 * \tau_2}{E'_1 \vdash fst\ x : \tau_1}\ \boxed{\tau_1 = \tau_3, \tau_2 = \tau_4}}{E'_1 \vdash g(fst\ x) : \tau}\ \boxed{\tau_1 = \tau_2}$$

On généralise le type.

$E_2 = \lfloor (f, \forall \tau_1, \tau . (\tau_1 \to \tau) * (\tau_1 * \tau_1) \to (\tau * \tau)) \triangleleft\ E_1 \rfloor$.

3

```
#let h x = f (copie,x);;
h : 'a * 'a -> ('a * 'a) * ('a * 'a) = <fun>
```

$$\dfrac{\dfrac{E_2' \vdash f : (\tau \to \tau_1) * (\tau * \tau) \to (\tau_1 * \tau_1) \quad (**)}{E_2' = \lfloor (x, \tau_x) \triangleleft\ E_2 \rfloor \vdash f(copie, x) : (\tau * \tau) * (\tau * \tau)} \quad \boxed{\tau = \tau_2, \tau_1 = \tau_2 * \tau_2, \tau_x = \tau * \tau}}{E_2 \vdash \texttt{fun}\ \ x \to f(copie, x) : (\tau * \tau) \to (\tau * \tau) * (\tau * \tau)}$$

avec (**) :

$$\dfrac{E_2' \vdash copie : \tau_2 \to (\tau_2 * \tau_2) \quad E_2' \vdash x : \tau_x}{E_2' \vdash (copie, x) : (\tau_2 \to (\tau_2 * \tau_2)) * \tau_x}$$

On généralise le type.
$E_3 = \lfloor (h, \forall \tau.(\tau * \tau) \to (\tau * \tau) * (\tau * \tau)) \triangleleft\ E_2 \rfloor$.

4

```
#let m x = h (x+1,x-1);;
m : int -> (int * int) * (int * int) = <fun>
```

$$\dfrac{\dfrac{E_3' \vdash f : (\tau * \tau) \to (\tau * \tau) * (\tau * \tau) \quad E_3 \vdash (x+1, x-1) : \texttt{int} * \texttt{int}}{E_3' = \lfloor (x, \tau_x) \triangleleft\ E_3 \rfloor \vdash h(x+1, x-1) : (\texttt{int} * \texttt{int}) * (\texttt{int} * \texttt{int})} \quad \boxed{\tau = \texttt{int}}}{E_3 \vdash \texttt{fun}\ \ x \to h(x+1, x-1) : \texttt{int} \to (\texttt{int} * \texttt{int}) * (\texttt{int} * \texttt{int})}$$

$E_4 = \lfloor (m, \texttt{int} \to (\texttt{int} * \texttt{int}) * (\texttt{int} * \texttt{int})) \triangleleft\ E_3 \rfloor$.

5 Expliquez la différence (du point de vue du typage) entre les deux expressions suivantes :

```
#let id = function x -> x;;
id : 'a -> 'a = <fun>

#if id true then id 1 else id 2;;
- : int = 1
```

et :

```
#let id = function x -> x;;
id : 'a -> 'a = <fun>

#(function f -> if f true then f 1 else f 2) id;;
Entrée interactive:
>(function f -> if f true then f 1 else f 2) id;;
>                                ^
Cette expression est de type int,
mais est utilisée avec le type bool.
```

Ces deux programmes calculent bien la même chose mais le premier est bien typé alors que le second ne l'est pas.

Voyons le typage du premier programme. `id` a le type polymorphe $\forall \tau.\tau \to \tau$. Ensuite :

$$\dfrac{\dfrac{E \vdash id : \tau_1 \to \tau_1}{E \vdash id\ true : \texttt{bool}} \quad \boxed{\tau_1 = \texttt{bool}} \qquad \dfrac{E \vdash id : \tau_2 \to \tau_2}{E \vdash id\ 1 : \texttt{int}} \quad \boxed{\tau_2 = \texttt{int}} \quad (**)}{E = \lfloor (id, \forall \tau.\tau \to \tau) \triangleleft\ E_0 \rfloor \vdash \texttt{if}\ \ id\ true\ \ \texttt{then}\ \ id\ 1\ \ \texttt{else}\ \ id\ 2 : \texttt{int}}$$

avec (**) :

$$\dfrac{E \vdash id : \tau_3 \to \tau_3}{E \vdash id\ 2 : \texttt{int}} \quad \boxed{\tau_3 = \texttt{int}}$$

La fonction polymorphe `id` a trois instances de type.

Par contre, la deuxième version est mal typée. En effet :

$$\dfrac{\dfrac{E_1 \vdash f : \tau_f \quad E_1 \vdash true : \mathtt{bool}}{E_1 \vdash f\ true : \mathtt{bool}} \quad \boxed{\tau_f = \mathtt{bool} \to \tau'_f} \quad \dfrac{E_1 \vdash f : \mathtt{bool} \to \tau'_f \quad E_1 \vdash 1 : \mathtt{int}}{E_1 \vdash f\ 1 :?}}{\dfrac{E_1 = \lfloor (f, \tau_f) \triangleleft E \rfloor \vdash \mathtt{if}\ f\ true\ \mathtt{then}\ f\ 1\ \mathtt{else}\ f\ 2 :?}{\dfrac{E \vdash (\mathtt{fun}\ f \to \mathtt{if}\ f\ true\ \mathtt{then}\ f\ 1\ \mathtt{else}\ f\ 2) :?}{E \vdash (\mathtt{fun}\ f \to \mathtt{if}\ f\ true\ \mathtt{then}\ f\ 1\ \mathtt{else}\ f\ 2)\ id :?}}}$$

Ce programme est mal typé parce que le type de f doit être à la fois égal à $\mathtt{bool} \to \mathtt{bool}$ et $\mathtt{int} \to \mathtt{int}$.
Remarque : Un tel programme pourrait être bien typé à condition de pouvoir affirmer que la fonction :

$$\mathtt{fun}\ f \to \mathtt{if}\ f\ true\ \mathtt{then}\ f\ 1\ \mathtt{else}\ f\ 2$$

a le type :

$$(\forall \tau . \tau \to \tau) \to \mathtt{int}$$

c'est-à-dire indiquant que le type de f doit être un schéma de type. Un tel type n'est pas définissable en Caml.