Département d'Informatique
LICENCE 2ème année
Lundi 26 novembre 2007

Interrogation écrite de Programmation Fonctionnelle

Durée : 2h. Les notes personnelles de cours sont autorisées. On doit utiliser le langage OCAML et on ne doit pas utiliser d'éléments impurs.

I Récursivité

1. Définir la fonction `toujours` qui, étant donnée une liste de booléens, retourne vrai si la liste ne contient que des vrais (et faux sinon).

   ```
   toujours [true; true; true; true] = true
   toujours [true; true; false; true] = false
   ```

2. Définir la fonction `parfois` qui, étant donnée une liste de booléens, retourne vrai si la liste contient au moins un vrai (et faux sinon).

   ```
   parfois [false; true; false; false] = true
   parfois [false; false; false; false] = false
   ```

II Algorithme de typage

Dérouler l'algorithme vu en Cours et en TD pour trouver le type de la fonction `f` définie par : `let f = function x -> function y -> x (x y) + 1;;` (On remplira 3 colonnes : une pour l'ensemble $\mathcal{H}$ des hypothèses, une pour l'ensemble $\mathcal{E}$ des équations et une pour l'ensemble $\mathcal{T}$ des expressions à typer).

III Types algébriques

1. Définir le type `expr_bool` des *expressions booléennes*. Par définition, une expression booléenne est :
   - soit la notation d'une valeur booléenne (vrai ou faux),
   - soit la disjonction (ou logique) de deux expressions booléennes,
   - soit la négation (non logique) d'une expression booléenne,
   - soit une variable booléenne dont le nom est une lettre de l'alphabet (de type `char`).

   (On utilisera respectivement les constructeurs `Val`, `Ou`, `Non` et `Var`.)

2. Écrire en OCAML l'expression booléenne

$$((\neg A) \vee B) \vee (\text{faux} \vee A)$$

où A et B sont des variables, $\vee$ est le symbole du "ou logique" et $\neg$ est le symbole du "non logique".
Dans la suite, on nomme `exemple`, cette valeur de type `expr_bool`.

3. On appelle *environnement*, une liste de couples (nom de variable, booléen) (de type `(char * bool) list`) : un environnement associe une valeur (de type `bool`) à tout nom de variable (de type `char`).
Écrire une fonction `eval` qui, étant donnés une expression booléenne (de type `expr_bool`) et un environnement, retourne la valeur (de type `bool`) de l'expression booléenne.

```
eval exemple [('A',true);('B',false)] = true
```

(On définira une fonction intermédiaire `val_of` qui, étant donnés un nom de variable et un environnement, retourne la valeur associée à la variable dans l'environnement.) (On supposera que tous les noms de variables de l'expression booléenne figurent dans l'environnement.)

4. Écrire une fonction `cree_envs` qui, étant donnée une liste de noms de variable booléenne (de type `char list`), retourne la liste de tous les environnements possibles.

```
cree_envs ['A';'B'] =
  [[('A',false);('B',false)];[('A',true);('B',false)];
   [('A',false);('B',true )];[('A',true);('B',true )]]
```

(On définira une fonction intermédiaire `adjoint` qui ajoute un élément en tête de toutes les listes d'une liste de listes.)

5. Une expression booléenne est *valide* si et seulement si sa valeur est toujours vrai quelque soit la valeur des variables qu'elle contient.
Écrire une fonction `est_valide` qui, étant données une expression booléenne et une liste de noms des variables, retourne un booléen indiquant si l'expression est valide ou non.

```
est_valide exemple ['A';'B'] = true
```

(On utilisera la fonction `toujours` du **I** 1.) (On supposera que la liste donnée en argument est la liste des noms de variables de l'expression booléenne.)